Windows Phone: Desenvolva para a nova plataforma móvel da Microsoft

Clube Delphi

FireMonkey e aplicações comerciais
Utilize animações visuais

ISSN 1517990-7

Ano 12 :: Edição 137 :: R$ 11,90

DevMedia

FastReport no Delphi XE2

Crie relatórios de forma ágil

Boas Práticas – Orientação a Objetos
Persistência de objetos no Delphi XE 2 com DORM

Boas Práticas – Orientação a Objetos
Padrões de projeto: Factory Method e Template Method

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

Sumário

Delphi

Delphi — **06 – FastReport no Delphi XE 2**
[*Giuliano Scombatti Pinto*]

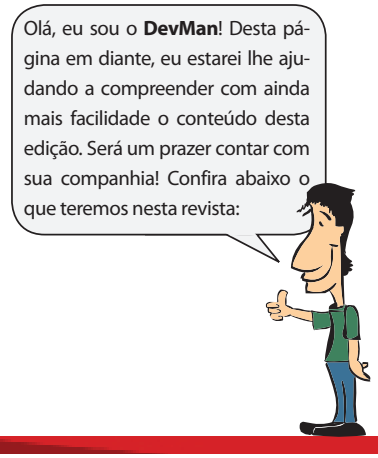
Delphi, Novidades — **14 – Animações em aplicações comerciais**
[*Giuliano Scombatti Pinto*]

Delphi, Novidades — **22 – Aplicações Windows Phone no Prism XE2**
[*Fabrício Hissao Kawata*]

Boas Práticas

Boas Práticas — **30 – Desenvolvimento OO no Delphi XE 2**
[*Paulo Quicoli*]

Boas Práticas — **40 – Padrões de Projeto: Factory Method e Template Method**
[*Renato Groffe*]



Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

www.devmedia.com.br/renovacao ou (21)3382-5038

DevMedia

Clube Delphi

Ano 12 - 137ª Edição 2012 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral

Paulo Quicoli
pauloquicoli@devmedia.com.br

Editor Técnico

Leandro Daniel
contato@leandrodaniel.com

Equipe Editorial

Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Guinther Pauli
guintherpauli@gmail.com

Jornalista Responsável

Kaline Dolabella - JP24185

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/mancad
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

EDITORIAL

É incontestável que a versão XE 2 do Delphi está recheada de novidades. Muitos podem pensar que elas se concentram apenas no framework FireMonkey, mas não é só isso. Nesta edição o colega Giuliano mostra uma introdução ao gerador de relatórios FastReport que agora faz parte integrada do Delphi. Talvez o FastReport não seja um grande conhecido da comunidade Delphi, mas posso dizer que é uma ferramenta incrível, recentemente pude participar de uma palestra, com o gerente de produtos da FastReport, e nos foi apresentado o que essa ferramenta pode fazer. Realmente é ótimo tê-la disponível no IDE.

Agora que temos em mãos um framework capaz de criar um visual rico, mais conhecimento será exigido dos desenvolvedores, e nem sempre será um conhecimento diretamente relacionado à programação. Será exigido sim um conhecimento mais visual, algo de designers. Assim, o Giuliano também apresenta alguns conceitos de animações visuais para aplicativos comerciais. Assim, você poderá criar um visual mais fluído para sua aplicação.

O desenvolvedor Delphi sempre pode desenvolver tudo com o Delphi. E para aplicativos móveis isso não é exceção, mesmo para a nova plataforma da Microsoft, o Windows Phone. É importante frisar que este não é um aparelho celular da Microsoft, mas, um sistema operacional que veio para enfrentar outros. O Fabrício explica muito bem isso e como você, desenvolvedor Delphi, pode utilizar o Prism XE2 para abraçar uma fatia desse mercado que tende a crescer muito em 2012.

Por fim, nosso amigo Renato Groffe aborda dois padrões de projeto: Factory Method e Template Method. Quando se procura o desenvolvimento orientado a objetos é importantíssimo conhecer suas boas práticas para garantir um software de qualidade. Os padrões de projeto seguem essa premissa e por isso devem ser conhecidos e quando possível, aplicados. Além dos padrões de projeto existem outros conceitos que permeiam o mundo orientado a objetos no mundo real, estes outros são apresentados em um artigo de minha autoria, onde abordo também a persistência de objetos e como utilizar o DORM, um novo framework ORM, para salvar seus objetos em um banco relacional, além disso, mostro também como utilizar o LiveBindings para apresentar seus objetos em janelas.

Boa leitura!

Paulo Quicoli
pauloquicoli@gmail.com
Editor Geral
quicoli.wordpress.com
twitter.com/pauloquicoli

Assinatura
ClubeDelphi PLUS

DevMedia

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8000 artigos, além de palestras e vídeos com formações completas.

TODAS as vídeo aulas que você sempre quis ver a partir de

R$ 1,20

ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!

Saiba mais sobre o Sistema de Créditos!

www.devmedia.com.br/creditos

DevMedia

FastReport no Delphi

FastReport 4 e soluções alternativas

Resumo DevMan

De que trata o artigo:

Este artigo apresenta uma introdução de como integrar as aplicações da plataforma VCL desenvolvidas no Delphi XE 2 com o FastReport 4, uma poderosa e sólida ferramenta de geração de relatórios integrada ao Delphi nesta versão, apresentando também uma solução alternativa para criar relatórios através do FireMonkey.

Em que situação o tema é útil:

Elaborar relatórios baseados no FastReport 4, presente na versão Delphi XE 2 e também disponível para versões anteriores, utilizando o editor visual da ferramenta, facilitando na criação, organização e visualização dos mesmos, bem como uma alternativa para "exportação de dados" a partir de aplicações baseadas em FireMonkey.

FastReport no Delphi XE2:

Atualmente a elaboração de relatórios em aplicações comerciais além de obrigatória, se tornou um grande diferencial, uma vez que os softwares podem conter relatórios de controle, financeiros, gerenciais, estatísticos, estratégicos, entre tantos outros, fazendo com que o usuário final deseje ou necessite obter a aplicação. O FastReport é uma ferramenta para a elaboração dos mais diversos tipos de relatório, possuindo um editor gráfico que permite a criação dos mesmos de uma maneira bem rápida e simples, aumentando assim a produtividade, utilizando também o famoso recurso arrastar e soltar, possibilitando ao desenvolvedor a criação de relatórios visualmente mais refinados com informações que podem ser facilmente destacadas e obtidas direto de DataSets inseridos em formulários do Delphi. Neste artigo é apresentada uma simples implementação de um relatório de Vendas para a familiarização com os recursos do FastReport, bem como algumas opções e recursos interessantes presentes no Editor do mesmo e para finalizar, um simples exemplo de como exportar dados de aplicações FireMonkey para HTML.

O Delphi XE 2 trouxe uma série de novidades que deram um novo "gás" para o desenvolvimento de aplicações em Delphi. A nova plataforma de desenvolvimento FireMonkey tem roubado a cena quando o assunto é a nova versão da ferramenta. Entretanto, apesar da FireMonkey possuir um futuro promissor, é uma nova tecnologia, portanto ainda não possui um vasto número de componentes adicionais e todas as funcionalidades necessárias para uma solução comercial completa, como é o caso de uma ferramenta específica para a geração de relatórios, afinal todas as disponíveis são voltadas para a plataforma VCL e vale a pena lembrar que a FireMonkey não é compatível com a mesma. Sendo assim, o desenvolvedor pode criar rotinas para desenvolver relatórios em HTML (ou outro formato) e utilizá-las na FireMonkey enquanto não há um suporte completo para esta tarefa, uma vez que relatórios são mais que essenciais em uma aplicação comercial por permitirem ao usuário final uma análise detalhada sobre sua empresa. Por outro lado, a VCL também recebeu atualizações significativas nesta versão.

Ao instalar o Delphi XE 2, o desenvolvedor conta também com a instalação

XE2

Giuliano Scombatti Pinto

do FastReport Embarcadero Edition (Versão 4). O FastReport é um componente adicional que permite a criação, edição, pré-visualização e impressão de relatórios de uma maneira eficiente e rápida, possuindo também um editor visual para a elaboração dos mesmos. Possui diversas versões como basic, standard, enterprise e professional, sendo também encontrado para edições anteriores do Delphi e outras plataformas de desenvolvimento, como é o caso da .NET. Possui um histórico sólido, uma vez que foi criado há mais de 10 anos e é utilizado por milhares de desenvolvedores ao redor do mundo. Vale a pena evidenciar que o FastReport não é uma ferramenta gratuita, portanto se o desenvolvedor desejar obter outras versões do mesmo, deverá efetuar sua compra, contudo possui versões de demonstração que podem ser facilmente obtidas (ver seção links).

Com relação aos recursos presentes no FastReport, além da interface avançada de desenvolvimento de relatórios, oferece suporte a mestre-detalhe, cache de grandes relatórios, herança, exportação para vários formatos (BMP, JPEG, HTML, PDF, TIFF, GIF, etc), encriptação, código de barras, entre outros recursos. Vale à pena evidenciar também que a versão presente no Delphi XE 2 não é a mais completa, onde recursos como bandas verticais, datasets internos, componentes para web, entre outros, não estão presentes.

Em meio a tantas novidades não se pode deixar de comentar também sobre outras ferramentas disponibilizadas em versões anteriores no que diz respeito à elaboração de relatórios, como é o caso do RaveReports, que também pode ser instalado em conjunto com o Delphi XE 2.

FastReport e suas Características

Como mencionado anteriormente, o FastReport para o Delphi XE 2 é uma ferramenta sólida para a geração de relatórios. Para integrar um relatório baseado em FastReport com um formulário Delphi, basta utilizar o objeto TfrxReport presente na paleta de componentes. O componente TfrxReport disponibiliza várias funcionalidades para que o desenvolvedor possa elaborar seus relatórios utilizando um editor constituído de uma interface gráfica amigável, permitindo também o carregamento e impressão do mesmo. Tal editor pode facilmente ser aberto através de um clique duplo sobre o objeto TfrxReport adicionado ao formulário.

Um outro componente útil é o TfrxDesigner, que também pode ser adicionado a um formulário, porém não está presente em todas as versões do FastReport. O TfrxDesigner é um componente que permite que o usuário possa alterar a aparência do relatório em tempo de execução, bem como editá-lo.

A interface gráfica do editor do FastReport poder ser facilmente personalizada através do menu View/Options, possibilitando a definição da unidade de medida (pixels, centímetros, etc), propriedades da grade exibida no relatório, definir a fonte utilizada no editor entre outras opções. Ele também possui uma janela que permite alterar as propriedades gerais do relatório, como a opção Print if empty que ativa e desativa a construção de relatórios em branco. Também existe a possibilidade de definir uma impressora que será utilizada como padrão através desta janela. Um recurso interessante que deve ser destacado é a utilização de herança nos relatórios, como já mencionado anteriormente, e que pode ser aplicado através da aba Inheritance desta janela, selecionando o relatório ao qual se deseja herdar. Para abrir a janela Report Settings que contém todos os recursos mencionados, basta acessar o menu Report/Options.

Outro passo importante na definição dos relatórios é o ajuste das propriedades da página. Tal ajuste pode ser efetuado através do menu File/Page Settings. Nesta opção o desenvolvedor pode definir o tamanho da página, o tipo do papel, a orientação (Retrato/Paisagem), as margens, etc. Outro ponto importante da janela Page Options é a definição do número de colunas da página, presente na aba Other Options.

Um objeto extremamente importante na elaboração de relatórios com o FastReport é o TfrxDBDataSet, presente na aba FastReport 4.0. O TfrxDBDataSet é um componente de conexão que permite a ligação entre uma Tabela (um objeto TSimpleDataSet por exemplo) ou um TDataSource presente em um formulário Delphi e o FastReport, sendo assim, se optar por utilizar um DataSet, com exceção do uso de Mestre-Detalhe, o desenvolvedor não necessita inserir componentes do tipo TDataSource para a conexão, em outras palavras, a conexão é direta entre o TSimpleDataSet e o TfrxDBDataSet. Tal componente de interligação é compatível com a maioria dos DataSets do Delphi (BDE, ADO, IBX, etc).

O FastReport possui uma diversidade de componentes que podem ser adicionados ao relatório, bandas, rótulos, imagens, dentre outros. Uma curiosidade na elaboração de relatórios com o FastReport é que o objeto Text pode ser usado para se obter outras funcionalidades além da exibição de um texto, como a data atual ou a página, através da utilização de [DATE] e [PAGE]. Exemplo: Data Atual: [DATE], Página: [PAGE]. Também pode ser utilizado para a exibição de um texto qualquer seguido de um campo de uma Tabela (DataSet), através da forma: Texto desejado[frxDBDataSet."Nome do Campo"]. Exemplo: Nome do Cliente: [frxDBDataSet1."NOME_CLIENTE"].

Como a maioria das ferramentas de relatórios, o desenvolvedor pode adicionar objetos Band (recipientes que permitem uma divisão e agrupamento dos obejtos) para organizar a estrutura do seu relatório, através do botão Insert Band do Editor. Evidentemente que ao utilizar os objetos band, o desenvolvedor deve seguir uma determinada ordem para a inserção das mesmas, ou seja, uma banda do tipo Page Header deve ser inserida antes de uma banda Master Data por exemplo. Ao inverter as ordens das bandas, o FastReport poderá gerar algumas mensagens de erro no momento em que o relatório for executado pela aplicação. Também existem objetos bands destinados a impressão dos dados obtidos através das tabelas ou consultas, nomeados DataBands. Neste caso, ao inserir um band do tipo DataBand é aberta uma janela onde o desenvolvedor deve selecionar qual o objeto TfrxDBDataSet que tal band esta relacionado para que os registros possam ser exibidos. Lembrando que a utilização do recurso Mestre-Detalhe nas bandas de dados do FastRport deve ser definida no próprio DataSet e DataSource do formulário Delphi. A **Tabela 1** demonstra os principais tipos de bands do FastReport.

Contudo, o FastReport suporta diversos recursos como variáveis, figuras, divisão do formulário em diversas colunas, entre outros.

FastReport na prática

Neste exemplo o FastReport será utilizado em uma aplicação VCL. Em um primeiro momento será criada uma base de dados com o Firebird nomeada "BASE.FDB", bem como uma tabela de Vendas e uma de Itens para que possam ser utilizadas nos dois exemplos contidos deste artigo. A **Listagem 1** demonstra o código SQL da tabela Vendas e a **Listagem 2** da tabela VendaItens.

Listagem 1. Código SQL da Tabela Vendas

```
CREATE TABLE VENDAS (
 ID_VENDA INTEGER NOT NULL,
  ID_CLIENTE INTEGER,
  NOME_CLIENTE VARCHAR(50),
  DATA TIMESTAMP,
  CONDICAO VARCHAR(1),
  VALOR_TOTAL NUMERIC(9,2),
  CONSTRAINT PKY_VENDA PRIMARY KEY(ID_VENDA)
);
```

Listagem 2. Código SQL da Tabela VendaItens

```
CREATE TABLE VENDAITENS (
  ID_VENDA INTEGER NOT NULL,
  ITEM INTEGER NOT NULL,
  ID_PRODUTO INTEGER,
  DESCRICAO_PRODUTO VARCHAR(50),
  QTDE NUMERIC(9,3),
  VALOR_UNIT NUMERIC(9,2),
  VALOR_TOTAL NUMERIC(9,2),
  CONSTRAINT PKY_VENDA_ITENS PRIMARY KEY(ID_VENDA, ITEM),
  CONSTRAINT FKY_VENDA FOREIGN KEY(ID_VENDA) REFERENCES VENDAS(ID_
VENDA) ON UPDATE CASCADE
);
```

Nota do DevMan

O recurso Mestre-Detalhe (Master-Detail) é baseado em um relacionamento entre uma tabela principal (mestre) e uma tabela filho (detalhe) através de um determinado campo, em outras palavras, um relacionamento um-para-muitos. Na prática seria a mesma coisa que ter uma tabela de vendas como principal e uma tabela de itens da venda como detalhe, onde a tabela mestre pode ter vários itens. Tal recurso se torna útil para filtrar ou agrupar as informações da tabela detalhe, exibindo apenas os registros que pertencem ao registro selecionado na tabela mestre, quer seja em um DBgrid, relatório, etc..

Nota do DevMan

Na área de informática, Tags (etiquetas) podem ser definidas como palavras-chave utilizadas para organizar várias informações. Normalmente são constituídas por <nome da tag + atributos>dados </nome da tag>. Note que são utilizados os símbolos "< >" para iniciar uma tag e os símbolos "</ >" para finalizar, devendo conter um mesmo nome para ambos, exemplo "<head></head>". Vale a pena ressaltar que uma determinada tag pode ter vários filhos (outras tags). A linguagem HTML (HyperText Markup Language) possui suas próprias tags, onde algumas devem ser dispostas em uma determinada ordem e outras podem ser utilizadas quantas vezes for desejado. As tags também são uma característica importante de documentos do tipo XML (Extensible Markup Language), muito utilizada atualmente na Internet e para armazenamento de informações. O interessante da linguagem XML é que o desenvolvedor pode criar suas próprias tags, elaborando uma estrutura de acordo com a necessidade do projeto.

Objeto Band	Descrição
Page Header	Recipiente exibido no topo de todas as páginas do relatório.
Page Footer	Recipiente exibido no rodapé de todas as páginas do relatório.
Report Title	Recipiente exibido apenas na primeira página, abaixo do Page Header, se o mesmo estiver presente.
Header	Recipiente utilizado para exibir alguma informação antes de um DataBand por exemplo, como o título dos campos de uma tabela de um Master Data.
Footer	Recipiente utilizado para exibir alguma informação depois de um DataBand por exemplo.
Master Data	Recipiente utilizado para exibir os registros de uma tabela mestre.
Detail Data	Recipiente utilizado para exibir os registros de uma tabela detalhe, podendo ser utilizada para o recurso Master-Detail por exemplo.
SubDetail Data	Recipiente utilizado para exibir os registros de uma tabela detalhe (outro nível de detalhamento).
Report Summary	Recipiente exibido no final do Relatório, normalmente utilizado para elaboração de um resumo.
Group Header	Recipiente que permite o agrupamento de bandas através de um determinado valor em comum. Ao ser inserido é solicitado um DataSet e um campo para o agrupamento.

Tabela 1. Principais tipos de objetos band do FastReport

Com o banco e as tabelas criadas, para dar sequência ao exemplo será necessário popular ambas as tabelas com as informações para que o relatório possa ser testado futuramente. Então criamos um projeto VCL e para a conexão com o banco de dados Firebird será inserido um objeto do tipo TSQLConnection da aba dbExpress no formulário. A propriedade ConnectionName deverá ser definida para FBConnection, bem como em Params, a propriedade Database para C:\Relatorios\BASE.FDB, ou seja, o endereço da base de dados. Também é interessante definir LoginPrompt como False para que não seja exibida a tela de Login. Agora dois objetos do tipo TSimpleDataSet e um do tipo TDataSource devem ser inseridos para o acesso as tabelas de Vendas e VendaItens. O TSimpleDataSet que conterá os dados da tabela Vendas foi renomeado para sdsVendas, sua propriedade Connection definida como SQLConnection1 (objeto de conexão com o banco de dados) e em DataSet.CommandText o seguinte código SQL: SELECT * FROM VENDAS. Para concluir, deve-se adicionar todos os campos da tabela efetuando um clique duplo no sdsVendas, um clique com o botão direito e selecionando a opção Add all Fields, para que todos os campos da tabela sejam adicionados.

O mesmo processo deverá ser repetido com o outro TSimpleDataSet, porém definindo seu nome como sdsVendaItens e em DataSet.CommandText adicionando o seguinte código SQL: SELECT * FROM VENDAITENS. O objeto TDataSource também deve ser configurado alterando a propriedade DataSet para seu respectivo objeto, no caso o sdsVendas. Também teve seu nome alterado para dsVendas. Posteriormente, a propriedade MasterSource do sdsVendaItens deve ser alterada para dsVendas (objeto TDataSource interligado ao sdsVendas), bem como em MasterFields, interligado os campos ID_VENDA de ambos os SimpleDataSets para utilizar o recurso Mestre-Detalhe. Com a base para o relatório criada, o próximo passo é inserir um objeto do tipo TfrxReport da aba FastReport 4.0 para dar início a elaboração de um relatório de vendas e seus devidos itens. Também deverão ser inseridos dois objetos do tipo TfrxDBDataSet para a conexão do FastReport com os DataSets do formulário. O frxDBDataSet1 foi renomeado para frxdbVendas nas propriedades Name e UserName, e a propriedade DataSet definida como sdsVendas. O frxDBDataSet2 também deve ter as mesmas alterações, porém com suas propriedades definidas para o sdsVendaItens. Para abrir o editor gráfico do FastReport e dar início a elaboração gráfica do relatório, basta dar um clique duplo no objeto frxReport1 criado anteriormente. Agora, para completar a ligação das Tabelas com o FastReport, basta utilizar a opção Data, presente no menu Report do editor e selecionar ambos objetos de interligação (frxdbVendaItens e frxdbVendas). A **Figura 1** demonstra a janela de Seleção dos DataSets.

Após adicionados os DataSets, os campos obtidos das tabelas aparecerão na janela Data Tree do Editor. A **Figura 2** demonstra os campos obtidos.

O título do relatório é criado com a inclusão da banda ReportTitle. A **Figura 3** demonstra o botão Insert Band e a opção Report Title, bem como as demais opções do mesmo.

Figura 1. Janela de Seleção de DataSets do FastReport

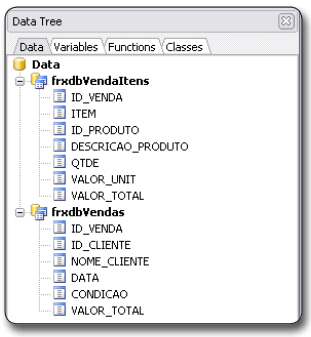


Figura 2. Campos obtidos das tabelas

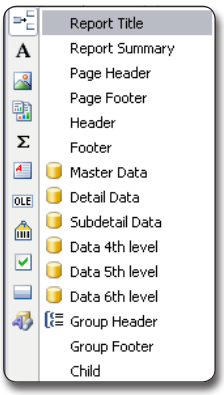
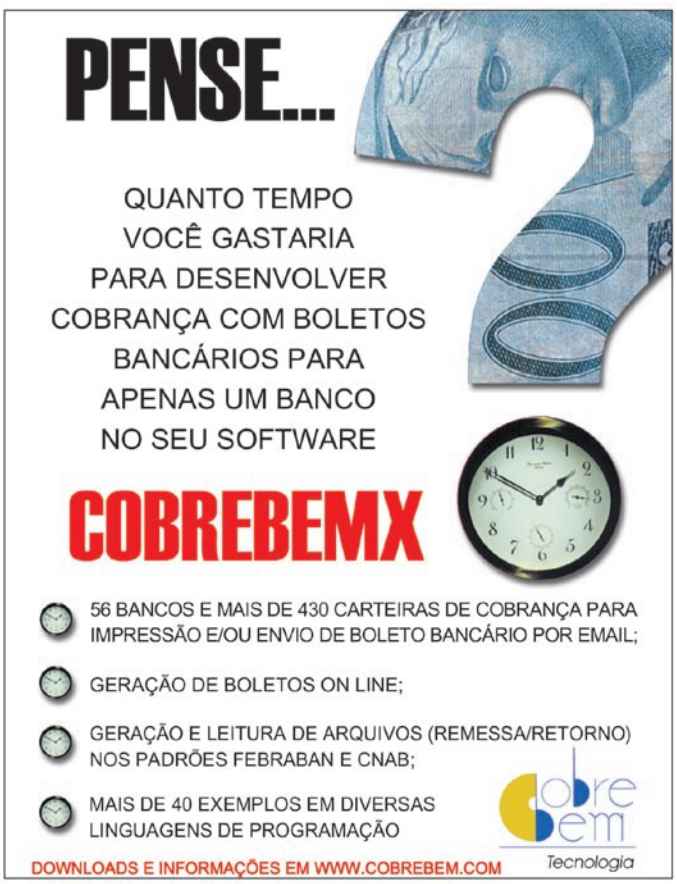


Figura 3. Botão Insert Band e suas opções



Posteriormente deve ser inserido um TextObject que permite a exibição de um texto no relatório (de forma parecida como um TLabel) devidamente preenchido com o título do relatório. Um objeto do tipo Header também deverá ser inserido através do botão Insert Band para conter a descrição dos campos da tabela de Vendas, onde evidentemente deverão ser adicionados os rótulos dos mesmos. Para dar continuidade basta inserir um objeto do tipo Master Data, também do Insert Band. Entretanto, ao inserir um objeto do tipo Master Data, será solicitado que o desenvolvedor selecione qual é o DataSet Mestre, como já mencionado anteriormente. A **Figura 4** demonstra a janela exibida ao inserir um Master Data.

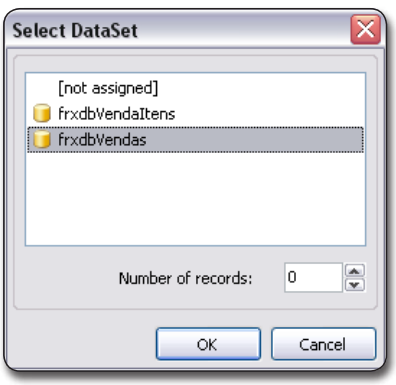


Figura 4. Janela de Seleção do DataSet Mestre

Dando sequência, basta inserir um novo objeto do tipo Detail Data, onde o objeto frxdbVendaItens deve ser selecionado para a exibição dos itens da venda no estilo Mestre-Detalhe. Também pode ser inserido um bando do tipo Footer, para servir de rodapé para cada registro de venda. Concluída a criação das bandas, é necessário inserir seus devidos campos de dados.

Uma das grandes facilidades do FastReport é o recurso de arrastar/soltar, permitindo assim que o desenvolvedor arraste os campos presentes na janela Data Tree diretamente para o relatório, evidentemente criando automaticamente a ligação entre ambos. Também foram inseridos dois objetos do tipo Text no topo do relatório, um contendo o texto "Data de Emissão: [DATE]" para a exibição da data atual, e um o texto "Página: [PAGE]" para a exibição do número da página. No final do projeto, o relatório de vendas terá a aparência da **Figura 5**.

Um outro recurso que pode ser utilizado se for da vontade do desenvolvedor, é alterar a cor de fundo dos objetos do relatório para uma melhor visualização, bastando apenas selecionar o objeto (um Text por exemplo) e alterar a sua propriedade Color.

Concluído o Projeto, o mesmo deverá ser salvo em um diretório para que possa ser carregado futuramente pela aplicação, evidentemente é interessante gravar o arquivo na mesma pasta do executável. Voltando ao Delphi, no formulário do exemplo criado deverá ser inserido um botão para carregar o relatório no momento em que o usuário clicar. A **Listagem 3** demonstra o código que pode ser executado para abrir a pré-visualização do relatório e a geração do mesmo no evento OnClick do Botão.

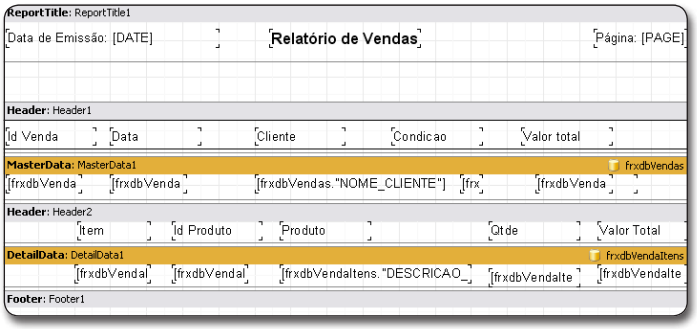


Figura 5. Aparência do relatório de Vendas

Listagem 3. Código do Botão responsável por abrir o relatório

```
procedure TForm1.Button1Click(Sender: TObject);
var
 Diretorio: string;
begin
 sdsVendas.Open;
 sdsVendaItens.Open;
 Diretorio := ExtractFilePath(Application.ExeName);
 with frxReport1 do
 begin
  LoadFromFile(Diretorio + 'vendas.fr3');
  ShowReport(true);
 end;
end;
```

Como pode ser observado no código da **Listagem 3**, primeiramente os DataSets são abertos. Posteriormente é obtido o diretório atual do executável e seu valor armazenado na variável Diretorio. Para carregar o relatório no objeto frxReport1 à partir do diretório é utilizada a função LoadFromFile(). Para concluir, o procedimento ShowReport() é utilizado para a pré-visualização e consequentemente a impressão do relatório.

Relatórios com FireMonkey

Como já mencionado anteriormente, ainda não existem ferramentas completas direcionadas à geração de relatórios especificamente para a plataforma FireMonkey. Contudo, maneiras alternativas podem ser utilizadas para a elaboração de um relatório através da mesma, ou seja, criar o relatório manualmente e exportá-lo para um arquivo TXT, HTML, entre outros formatos, e posteriormente executar uma aplicação para abrir o mesmo e possibilitar sua impressão. Quando o ambiente de desenvolvimento é o Windows, tal elaboração se torna mais fácil, ou seja, os códigos utilizados para a geração de relatórios em HTML e posteriormente o carregamento de um programa para exibição, por exemplo, são os mesmos que seriam utilizados na VCL, contudo vamos mostrar como realizar isso com o FireMonkey. Iniciamos uma nova aplicação FireMonkey e vamos realizar uma conexão com um banco de dados. Os componentes utilizados para a conexão com o banco de dados e tabelas serão os mesmos descritos no início deste artigo, bem como suas propriedades (1 TSQLConnection e 2 TSimpleDataSet). Considerando que os componentes estão devidamente

interligados, basta inserir um botão na aplicação FireMonkey e no evento OnClick do mesmo inserir o código da **Listagem 4** para a elaboração de um relatório baseado em HTML.

Como pode ser observado, o primeiro passo é a abertura dos Data Sets e o posicionamento para o primeiro registro. Posteriormente é utilizada uma variável nomeada html que armazenará todo o código HTML para o relatório. No primeiro laço for, é gerada uma célula para cada campo da tabela venda através das tags <tr> e <td>. Posteriormente são inseridos os registros da tabela venda, bem como todos os seus itens, onde para cada registro é gerada uma célula. No final do código, é utilizado o procedimento AssignFile() para a criação de um arquivo texto. Posteriormente um Rewrite() para reescrever no arquivo (caso já exista) e um writeln() que escreve todo o conteúdo da variável html. Por fim o arquivo é fechado através de CloseFile(). Para concluir, a função ShellExecute() é utilizada para executar uma aplicação Windows externa, neste caso o navegador de internet, bem como o arquivo que deverá ser aberto e o estado da janela (Maximizada). Vale a pena lembrar que é necessário adicionar as Units Winapi.ShellAPI e Winapi.Windows para o uso desta função.

Como é possível observar neste exemplo, para gerar um relatório baseado em HTML existe a necessidade de certo conhecimento sobre a linguagem, principalmente se o desenvolvedor desejar formatar a aparência do mesmo, podendo obter um relatório visualmente rico.

Uma outra alternativa para gerar relatórios com HTML e Fire Monkey é utilizar o objeto TDataSetTableProducer da Unit Web.DBWeb. O objeto TDataSetTableProducer permite a criação de uma tabela HTML à partir de um DataSet, sem que exista a necessidade de vastos códigos. Dando sequência ao projeto, foi inserido um novo botão e o código da **Listagem 5** foi inserido no evento OnClick do mesmo para gerar um relatório à partir de um DataSet com o TDataSetTableProducer.

Como é possível observar, o código não exige tantos conhecimentos na linguagem HTML e permite gerar um relatório em HTML de uma maneira muito mais ágil. É evidente que ao trabalhar com vários relatórios em HTML, o ideal é criar um procedimento e passar os DataSets como parâmetros para aumentar a produtividade.

Conclusão

O FastReport, agora integrado ao Delphi XE 2, aumentou as opções para a elaboração de relatórios para os desenvolvedores. Um de seus grandes diferenciais é o editor visual com vários recursos, permitindo a elaboração de relatórios de uma maneira mais rápida e visualmente rica, consequentemente oferecendo uma maior produtividade no projeto. Tal parceria entre a FastReports e a Embarcadero tornou-se mais um atrativo para os desenvolvedores, uma vez que o FastReport é uma ferramenta proprietária e deve ser comprada para que possa ser utilizada em outras versões.

O FastReport 4 integrado ao Delphi, está disponível apenas para a plataforma VCL, entretanto a empresa já anunciou que deve desenvolver versões para a FireMonkey. Contudo a FastReport lançou recentemente uma nova versão do FastScript, a 1.98 que oferece suporte a FireMonkey, inclusive contendo exemplos indicando os primeiros passos para a elaboração de uma futura ferramenta para elaboração de relatórios para a plataforma FMX.

Listagem 4. Código do Botão responsável por gerar o relatório em HTML

```
procedure TForm1.Button1Click(Sender: TObject);
var
 html: widestring;
 Arquivo: TextFile;
 i: Integer;
begin
 sdsVendas.Open;
 sdsVendaItens.Open;
 sdsVendas.First;
 html := '<html>';
 html := html + '<head>';
 html := html + '<title>RELATÓRIO DE VENDAS</title>';
 html := html + '<style>';
 html := html + 'h1{font: bold 20px arial; text-decoration: underline;}';
 html := html + 'a{font: bold 10px arial; text-decoration: none;}';
 html := html + '.tditem{padding-left: 20px; border-top: 1px solid #BBB;}';
 html := html + '.tdvenda{border-top: 1px solid #000;}';
 html := html + '</style>';
 html := html + '</head>';
 html := html + '<body>';
 html := html + '<div align="center">';
 html := html + '<h1>RELATÓRIO DE VENDAS</h1>';
 html := html + '<table border="0" cellpadding="0" cellspacing="0">';
 html := html + '<tr>';
 for i := 0 to sdsVendas.FieldCount -1 do
  html := html + '<td class="tdvenda">' + sdsVendas.Fields[i].DisplayName + '</td>';
 html := html + '</tr>';
 while not sdsVendas.eof do
 begin
  html := html + '<tr>';
  for i := 0 to sdsVendas.FieldCount -1 do
   html := html + '<td class="tdvenda">' + sdsVendas.Fields[i].AsString + '</td>';
  html := html + '</tr>';
  html := html +'<tr><td colspan="'+ IntToStr(sdsVendas.FieldCount) +'">'+
          '<table border="0" width="100%" cellpadding="0" cellspacing="0">';
  html := html + '<tr>';
  for i := 0 to sdsVendaItens.FieldCount -1 do
   html := html + '<td class="tditem">' + sdsVendaItens.Fields[i].DisplayName + '</td>';
  html := html + '</tr>';
  while not sdsVendaItens.Eof do
  begin
   html := html + '<tr>';
   for i := 0 to sdsVendaItens.FieldCount -1 do
    html := html + '<td class="tditem">' + sdsVendaItens.Fields[i].AsString + '</td>';
   html := html + '</tr>';
   sdsVendaItens.Next;
  end;
  html := html + '</table></td></tr>';
  sdsVendas.Next;
 end;
 html := html + '</table>';
 html := html + '</div>';
 html := html +'</body></html>';
 AssignFile(Arquivo, 'C:\Relatorio.htm');
 Rewrite(Arquivo);
 Writeln(Arquivo, html);
 CloseFile(Arquivo);
 ShellExecute(0, 'open', 'C:\Relatorio.htm',nil,nil, SW_SHOWMAXIMIZED);
end;
```

Listagem 5. Código utilizado para gerar uma Tabela HTML à partir de um TDataSet TableProducer

```
procedure TForm1.Button2Click(Sender: TObject);
var
 tabela: widestring;
 tableProducer: TDataSetTableProducer;
 Arquivo: TextFile;
begin
 sdsVendas.Open;
 try
  tableProducer := TDataSetTableProducer.Create(nil);
  sdsVendas.First;
  with tableProducer do
  begin
   tableProducer.DataSet := sdsVendas;
   TableAttributes.CellSpacing := 0;
   TableAttributes.CellPadding := 0;
   TableAttributes.Border := 1;
  end;
  tabela := tableProducer.Content;
  AssignFile(Arquivo, 'C:\Relatorio2.htm');
  Rewrite(Arquivo);
  Writeln(Arquivo, tabela);
  CloseFile(Arquivo);
  ShellExecute(0, 'open', 'C:\Relatorio2.htm',nil,nil, SW_SHOWMAXIMIZED);
 finally
  FreeAndNil(tableProducer);
 end;
end;
```

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Download do FastReport 4 e seus Manuais
http://www.fast-report.com/en/download/fast-report-4-download.html

Informações sobre o FastScript para a FireMonkey
http://www.fast-report.com/en/download/fast-script-download.html

DevSpace do Autor
http://www.devmedia.com.br/space.asp?id=283274

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Seu **conhecimento vale muito** na DevMedia

Publique artigos e vídeos gratuitamente no site e participe do programa **PayContent.**

Sua participação pode ser premiada: quanto mais você postar, mais chances de ganhar.

Para saber mais acesse:
www.devmedia.com.br/paycontent

DevMedia

Animações em aplica
FireMonkey na prática

Resumo DevMan

De que trata o artigo:

Este artigo demonstra como utilizar as animações visuais da nova ferramenta do Delphi XE 2, a FireMonkey, apresentando também uma idéia de como empregá-la em aplicações comerciais.

Em que situação o tema é útil:

Desenvolver uma aplicação comercial baseada em FireMonkey utilizando seus recursos visuais, como simples animações, para obter uma interface final mais rica e agradável ao usuário, bem como proporcionar ao desenvolvedor uma idéia geral do funcionamento das animações da nova tecnologia.

Animações em aplicações comerciais:

A nova ferramenta FireMonkey (FMX) integrada ao Delphi tem demonstrado que é uma plataforma rica para o desenvolvimento de aplicações com poderosos efeitos gráficos 2D ou 3D, sem contar que é portável. Com a grande ascensão das tecnologias de desenvolvimento de Software nos últimos tempos, principalmente no quesito visual e interação com o usuário, ela vem como uma poderosa aliada que facilita a utilização de tais recursos para o desenvolvedor. Neste artigo, o foco é a utilização de animações simples e rápidas com oFireMonkey, disponíveis nativamente pela própria ferramentasem a utilização de componentes adicionais, para serem integradas em aplicações com um foco comercial. Também demonstra uma simples utilização de LiveBindings em conjuto com o dbExpress para acesso ao banco de dados Firebird.

O mercado de desenvolvimento de software tem ganhado várias ferramentas e tecnologias para o projeto de aplicações mais ricas em interface e interatividade. É evidente que com a popularização do recurso Touch Screen e da atual ascensão dos dispositivos portáteis como os Smartphones, tais tecnologias se tornem cada vez mais presentes no cotidiano dos desenvolvedores. Até certo tempo atrás, o programador Delphi que desejava ou necessitava desenvolver aplicações com interfaces mais ricas ou personalizadas para o ambiente Windows, precisava utilizar componentes de terceiros ou desenvolver seus próprios códigos, uma vez que a VCL e o próprio ambiente nativo da ferramenta não apresentavam tantos recursos para a manipulação da interface gráfica dos aplicativos. Com o surgimento da versão XE2 do Delphi novas tecnologias foram incorporadas à ferramenta, principalmente no quesito interface.

FireMonkey

O FireMonkey (FMX), que é uma nova plataforma para aplicações, tem atraído a atenção dos desenvolvedores. Ela é uma biblioteca de componentes gráficos que utiliza a GPU (Unidade de Processamento Gráfico) e também a CPU (Unidade Central de Processamento) dos computadores ou dos dispositivos portáteis, uma vez que é multi-plataforma, possibilitando o desenvolvimento de aplicações HD 3D ricas visualmente e de uma maneira rápida, com vários efeitos visuais. Ela possui uma camada de abstração de recursos, um motor GUI que cuida geração de caixas de texto, botões, janelas e demais componentes, o que possibilita sua portabilidade para

ções comerciais

Giuliano Scombatti Pinto

outros ambientes, uma vez que os mesmos ficam independentes do Sistema Operacional. Oferece suporte ao desenvolvimento de gráficos em 2D ou 3D, também utilizando formas primitivas, possuindo um motor de vetores poderoso. Também é constituída por um motor destinado a utilização de temas pré-definidos ou criados pelo próprio usuário através de um Editor. Suporta animações, que por sua vez, são calculadas em segundo plano através de uma Thread, apresentando um consumo mínimo de CPU e correção automática da taxa de frames (quadros), permitindo também a utilização e manipulação de vários tipos de imagens como Bitmap, JPEG, PNG, GIF, etc.

O FireMonkey e a VCL não podem ser utilizados em conjunto num mesmo projeto pois são incompatíveis. Vale a pena lembrar também que a VCL é uma biblioteca de componentes que é suportada apenas no ambiente Windows. Atualmente o FireMonkey é compatível com o ambiente Windows 32 ou 64 bits (a partir do XP SP2), Mac OS X e iOS, porém deverá oferecer suporte a outras plataformas futuramente.

O desenvolvimento 3D com o Delphi foi enfatizado com o FireMonkey. Utilizando um objeto do tipo TForm3D ou TViewPort3D, por exemplo, é possível adicionar vários objetos 3D primitivos, como um Cubo, e criar câmeras, efeitos de iluminação e animação na aplicação, como rotação e escala, bem como a utilização de materiais para as superfícies dos objetos. Também é importante evidenciar que ao integrar animações com aplicações comerciais, é uma boa prática levar em consideração o tempo de execução da animação, não utilizando efeitos demorados, afinal além de uma aparência agradável o usuário deseja ter um sistema que responda o mais rápido possível às suas requisições.

Entretanto o FireMonkey não é apenas uma plataforma destinada ao desenvolvimento de uma aplicação gráfica ou cheia de efeitos e animações, mas sim uma solução completa para aplicativos comerciais. Ela suporta o acesso aos bancos de dados através das ferramentas DataSnap e dbExpress, podendo o desenvolvedor utilizar recursos deLiveBinding que utiliza expressões para a interligação dos objetos através das suas propriedades.

Classes FireMonkey de animações

A plataforma FireMonkey disponibiliza recursos para que o desenvolvedor possa criar animações em sua aplicação como descrito anteriormente. Vale apena evidenciar que animações podem ser obtidas através da alteração das propriedades dos componentes da aplicação com relação ao tempo. Tais recursos podem ser inicializados de forma automática ou manual através de um evento por exemplo, assim como reiniciados ou efetuados de maneira inversa.

Nota do DevMan

Nos sistemas operacionais os programas são executados como processos. Uma Thread pode ser definida como um subprocesso pertencente a algum processo em execução, permitindo assim a execução de códigos em paralelo (segundo plano), possibilitando também a troca de informações entre ambas. Uma aplicação também pode conter várias Threads em execução compartilhando suas informações.

Nota do DevMan

O recurso de LiveBinding é um mecanismo que permite a ligação entre os objetos da aplicação por meio de expressões relacionais, também conhecidas como expressões de ligação. Diante disso um objeto pode estar interligado a outros objetos bem como a si mesmo. No Delphi, ao criar um novo LiveBinding através da propriedade LiveBindings e selecionada a opção Link to DBField, presente em diversos componentes, é criado um TBindingList, que permite que sejam visualizadas e alteradas as ligações presentes no formulário, facilitando o trabalho de ligação. Também é criado um TBindScopeDB para a definição de um DataSource para o DataBinding.

Quanto aos tipos de animações disponibilizados por esta plataforma, sãodivididos em três categorias: interpolações originadas por dois valores (a partir de um valor inicial até um valor final), interpolações originadas de vários valores e sem interpolação, neste último caso como se fosse uma apresentação de slides baseada em uma lista de imagens. Em animações gráficas,interpolaçãoé o termo utilizado para definir a criação automática de uma sequência de novos quadros tendo como base determinados valores, comoum valor inicial e um valor finalonde cadaquadro gerado depende do quadro anterior até que atinja o valor estabelecido como final, levando também em consideração um determinado tempo. Sendo assim, se a animação é uma rotação e são definidos os valores1 (inicial) e 5 (final), os quadros 2,3 e 4 serão gerados de uma maneira automática, evidentemente de acordo com o tempo definido.Tais interpolações desta plataforma são semelhantes às utilizadas pelo Flash, uma ferramenta mais voltada para o desenvolvimento de efeitos para a Web. A **Tabela 1** descreve os tipos de animações com interpolação originada por dois valores.

Já a**Tabela 2** demonstra as interpolações originadas de vários valores.

As animações podem ser anexadas a um objeto para que o mesmo seja animado assim como a qualquer subcomponente.

Subclasse	Descrição
TFloatAnimation	Permite a alteração de uma propriedade numérica como o valor dos eixos (x,y e z), opacidade e rotação, etc.
TRectAnimation	Permite a alteração dos cantos presentes na propriedade TBounds.
TColorAnimation	Permite a alteração de propriedades relacionadas á cor, definida através do vermelho, verde e azul, bem como valores alfa.
TGradientAnimation	Permite a mudança de cores de um objeto TGradient, um objeto constituído por uma determinada cor inicial que se mistura até chegar a uma outra cor (final).
TBitmapAnimation	Permite a transição de uma imagem bitmap para outra, resultando em um efeito fade por exemplo.

Tabela 1. Tipos de animações com interpolação originada por dois valores

Através do ObjectInspector as propriedades de animação são apresentadas através de um ícone de filme e, sempre que um objeto de animação é adicionado pelo desenvolvedor, a propriedade PropertyName do mesmo deve ser definida manualmente. Para que uma animação seja executada automaticamente na inicialização da aplicação, basta definir sua propriedade Enabled para True em tempo de desenvolvimento, e evidentemente, tal propriedade pode ser alterada para Verdadeiro em tempo de execução para que a mesma seja iniciada em um determinado momento da aplicação, através de um clique, por exemplo, bem como posteriormente definida como False para ser parada.

Subclasse	Descrição
TFloatKeyAnimation	Permite transições com base em uma lista de números.
TColorKeyAnimation	Permite transições com base em uma lista de cores.
TPathAnimation	Permite a modificação da posição de um objeto 2D, oferecendo a opção de rotação do mesmo durante o percurso.

Tabela 2. Tipos de animações com interpolação originada por vários valores

TFloatAnimation

O objeto TFloatAnimation da Firemonkey é utilizado para criar uma animação a partir da alteração do valor de uma determinada propriedade de um componente visual, onde tal propriedade deve ser informada. Também são informados o tipo de alteração e a duração da mesma.

Quando um objeto do tipo TFloatAnimation é utilizado para uma animação, se a propriedade StartFromCurrent é definida como True, permite que a animação seja iniciada a partir da posição atual do objeto e automaticamente desconsidera o valor presente em StartValue, utilizada para definir um valor fixo para o início. A propriedade StopValue pode ser preenchida com um valor para definir o momento que a animação será encerrada, se a propriedade Inverse estiver definida como False. Entretanto, uma animação também pode ser iniciada através de uma Trigger (gatilho), em outras palavras, quando o usuário interage com a aplicação através do mouse ou teclado, fazendo com que uma animação possa ser executada enquanto o mesmo mantém a interação, ou quando determinada propriedade esta ativada ou desativada. As propriedades fornecidas para serem utilizadas com triggers são: IsMouseOver, IsDragOver, IsFocused e IsVisible. Alguns controles possuem outras propriedades como IsPressed, IsChecked, IsSelected, IsExpanded e IsActive. Tais propriedades possuem um valor do tipo boolean para que o desenvolvedor possa testar seus valores e definir qual será a animação executada. Tais valores podem ser definidos na propriedade Trigger, para começar a animação, e TriggerInverse, para que a animação seja executada de maneira invertida.

A **Tabela 3** demonstra algumas das propriedades do objeto TFloatAnimation.

Nota do DevMan

O conceito de Triggers (gatilho) é bem conhecido pelos usuários de bancos de dados. Em resumo, permite determinar que um procedimento será executado quando certa ação for efetuada ou satisfeita. A utilização de efeitos baseados em Triggers noFireMonkey permite que um efeito como a alteração de uma imagem possa ser executado de acordo com os valores das propriedades de determinado objeto com relação àpropriedade trigger definida como resposta, por exemplo IsMouseOver = Verdadeiro. Sendo assim, as triggers são executadas com base em tais propriedades verificadas durante o processamento do componente. Todos as propriedades das triggers retornam um valor Verdadeiro ou Falso.

Propriedade	Definição
Duration	A duração da animação.
Enabled	Utilizado para iniciar a animação.
PropertyName	A propriedade do componente que deverá ser alterada.
StartFromCurrent	Determina que o valor inicial da animação é o valor definido atualmente na propriedade configurada.
StartValue	Determina qual o valor inicial, sendo ignorado quando StartFromCurrent é definido como verdadeiro.
StopValue	Determina o valor final.
Trigger	Define a execução de um animação de acordo com o valor obtido através das propriedades IsMouseOver, IsDragOver, IsFocused e IsVisible.

Tabela 3. Propriedades mais utilizadas do objeto TFloatAnimation

Alguns componentes visuais do FireMonkey como o TButton, já possuem um procedimento que também permite a criação de uma animação. Para tanto basta utilizar AnimateFloat(). Neste caso o objeto TFloatAnimation é instanciado pelo procedimento e liberado da memória após o término da animação. Sua estrutura é a seguinte: [Objeto].AnimateFloat([Nome da Propriedade],[Novo Valor],[Duração], TAnimationType.[Valor],

TinterpolationType.[Valor]). Um exemplo da sua utilização seria: Button1.AnimateFloat('Roation.Z', 500, 1, TAnimationType.atIn, TinterpolationType.itLinear), para obter um efeito de rotação no eixo Z. Entretanto, os parâmetros TAnimationType e TInterpolationType podem ser omitidos pelo desenvolvedor se desejado.

Como observado, para a utilização do procedimento Animate Float() alguns parâmetros podem ser informados para obter um efeito mais específico, como é o caso do TAnimationType e TInterpolationType.

O objetoTAnimationType descendente da classe FMX.Types define como será aplicada a interpolação da animação, em outras palavras, é possível especificar como o valor da propriedade definida é alterado á partir de seu valor inicial até o seu valor final. A **Tabela 4** demonstra os valores de TAnimationType que são aplicados no objeto TFloatAnimation.

Valor	Definição
atIn	Indica que a curva que será utilizada pelo TInterpolationType desta animação começa no valor inicial da propriedade.
atOut	Indica que a curva que será utilizada pelo TInterpolationType desta animação começa no valor final da propriedade seguindo até o valor inicial.
atInOut	Indica que a curva que será utilizada pelo TInterpolationType desta animação começa no valor inicial e no valor final da propriedade. (vgAnimationInOut)

Tabela 4. Valores do objeto TAnimationType

O objetoTInterpolationType descendente de FMX.Types define o tipo da propriedade de interpolação da animação, ou seja, a taxa em que o valor inicial de uma determinada propriedade será alterado até seu valor final durante o tempo especificado. A **Tabela 5** demonstra os tipos de TInterpolationType que podem ser aplicados a um objeto TFloatAnimation.

O primeiro parâmetro do procedimento AnimateFloat() define qual será a animação utilizada como descrito anteriormente.A **Tabela 6** demonstra algumas propriedades mais comuns que podem ser utilizadas com o mesmo.

Propriedade	Definição
Position	Permite criar uma animação baseada na posição, ou seja, permite mover um objeto, onde é definida uma subpropriedade (X,Y e Z) para determinar o eixo do movimento. Exemplo: Position.Y
Rotation	Permite criar uma animação baseada em rotação onde é definida uma subpropriedade (X,Y e Z) para determinar o eixo da mesma. Exemplo: Rotation.Y
Opacity	Efeito de opacidade normalmente conhecido como fade.

Tabela 6. Algumas propriedades utilizadas no AnimateFloat()

Além do procedimento AnimateFloat(), os componentes também possuem o procedimento AnimateFloatDelay(). O AnimateFloatDelay() tem o mesmo objetivo que o AnimateFloat() porém possui um parâmetro a mais (Delay ou Atraso) que faz com que o procedimento aguarde o tempo definido neste parâmetro para iniciar a execução da animação. Sua sintaxe é: [Objeto].AnimateFloatDelay([Nome da Propriedade],[Novo Valor],[Duração],[Delay ou Atraso],TAnimationType.[Valor], TinterpolationType.[Valor]).

O procedimento AnimateFloat() é muito útil para criação de animações e possui uma particularidade que deve ser levada em consideração quando se deseja combinar animações. Ao utilizar este tipo de procedimento, a animação é executada e antes que ela seja finalizada, os próximos trechos de código também serão executados, consequentemente se houverem outros procedimentos do tipo AnimateFloat(), as animações serão executadas praticamente ao mesmo tempo. Contudo, se o desenvolvedor deseja que as animações sejam executas em série ou que o próximo trecho de código a ser executado deva aguardar o término da animação, ele pode utilizar o procedimento AnimateFloatWait(), fazendo com que seja aguardada a execução e a finalização da animação para dar continuidade. Sua sintaxe é idêntica a do procedimento AnimateFloat().

TColorAnimation

O objeto TColorAnimation permite que seja alterada a cor de um determinado componente visual durante um determinado tempo

Valor	Definição
itLinear	Interpolação Linear, onde o valor da propriedade em questão é alterado de maneira constante durante o tempo de execução da animação. (vgAnimationLinear)
itQuadratic	Neste caso é aplicada a função quadrática durante o percurso da animação do valor inicial até o valor final.
itCubic	Interpolação cúbica (y = x** 3), onde o valor de inclinação é aumentado durante a execução de uma forma mais rápida que a função quadrática.
itQuartic	Interpolação da fórmula y = x** 4, onde o valor de inclinação também é aumentado mais rápido que a função quadrática.
itQuintic	Interpolação da fórmula y = x** 5, onde o valor de inclinação também é aumentado mais rápido que a função quadrática.
itSinusoidal	Interpolação da fórmula y = sin(x).
itExponential	Interpolação da fórmula y = e** x, onde o valor de inclinação também é aumentado mais rápido que a função quadrática.
itCircular	O trecho percorrido do valor inicial até o final equivale a um quarto de circulo.
itElastic	Neste caso o percurso não tem uma interpolação geométrica, onde o valor x se dirige a uma direção positiva e o y pode diminuir.
itBack	Neste caso o percurso não tem uma interpolação geométrica, onde o valor x se dirige a uma direção positiva e o y pode diminuir também.
itBounce	O percurso é feito através de curvas, gerando um efeito como uma bola quicando.

Tabela 5. Valores do objeto TInterpolationType

criando um efeito de animação. Ele é ligado a tal componente através da propriedade TColor.

O procedimento AnimateColor() funciona de uma maneira parecida com o AnimateFloat() descrito anteriormente. Ele está presente em alguns objetos visuais da FireMonkey e cria uma instância de um objeto TColorAnimation. Como seu próprio nome sugere, permite criar uma animação baseada em cor. Sua estrutura é a seguinte: [Objeto].AnimateColor([Nome da propriedade],[Cor],[Duração], TAnimationType.[Valor], TInterpolationType.[Valor]). Um exemplo da sua utilização seria Rectangle1.AnimateColor('Color.Fill', $FF000000, 2, TAnimationType.atInOut, TinterpolationType.itExponential), para obter um efeito de mudança de cor. Entretanto, os parâmetros TAnimationType e TinterpolationType também podem ser omitidos se desejado onde sua utilização é semelhante a da função AnimateFloat().

FireMonkey, Animações e Aplicações Comerciais

OFireMonkey será utilizado para o desenvolvimento de uma simples aplicação com um cadastro de clientes. Como mencionado anteriormente, quando animaçõessão integradas à aplicações comerciais é de total relevância definir durações curtas para os efeitos, evidentemente levando em consideração que o usuário pode estar"ansioso" para acessar uma outra função do sistema. O primeiro passo é criar um novo projeto FireMonkey HD Application.

Para aplicar um visual diferenciado na aplicação, será utilizado um estilo (tema). Através de um estilo é possível alterar o visual completo da aplicação, desde as cores até o formato dos objetos. O Delphi XE 2 já possui alguns estilos por padrão, adicionados durante o processo de instalação da ferramenta. Os estilos estão disponíveis no diretório C:\Documents and Settings\All Users\Documentos\RAD Studio\9.0\Styles. Contudo se o desenvolvedor preferir criar seu próprio estilo, basta carregar um padrão e alterá-lo. Para adicionar ou editar um estilo é necessário inserir um objeto do tipo TStyleBook no formulário. Com o objeto adicionado, basta dar um clique duplo no mesmo e o editor será aberto. Para dar início a um estilo personalizado, basta clicar em Load Default e serão carregados os visuais padrões dos objetos do Firemonkey, disponibilizando assim uma inteface para a edição de cada componente. Para utilizar um estilo pronto no formulário, basta clicar em Load e selecionar o mesmo acessando o diretório padrão citado anteriormente.

Neste exemplo foi selecionado o estilo Dark. Após selecionado ou editado o estilo, basta em clicar em Apply and Close. O próximo passo é selecionar o formulário e na propriedade StyleBook definir como StyleBook1, o objeto TStyleBook criado anteriormente que contém o estilo selecionado. Pronto, o formulário e os objetos que forem inseridos, como um botão, já estarão com um novo visual. Também é interessante alterar a propriedade Position do formulário para poScreenCenter, fazendo com que o mesmo fique centralizado na tela. Para dar sequência ao projeto, insere-se um TPanel com a propriedade Align configurada como alLeft, para que o mesmo oculpe toda a área esquerda do TForm. Também será inserido um TButton no Painel para futuramente exibido cadastro de clientes com uma animação e um objeto do tipo TMenuBar para criar um menu estilizado, devendo também definir sua propriedade align para alTop para oculpar toda a área superior do formulário. O objeto TMenuBar, assim como o TMainMenu, também possui um editor de menus e submenus. Para acessá-lo basta dar um clique duplo sobre o mesmo.

Neste exemplo foi criado um TMenuItem e definido em sua propriedade Text o valor Aplicação, bem como um Children (filho), neste caso um submenu, defindo como Sair. O código atribuído ao TMenuItem Sair no evento OnClick foi o Application.Terminate, para finalizar a execução da aplicação. A **Figura 1** demonstra o estilo visual da aplicação até o momento.

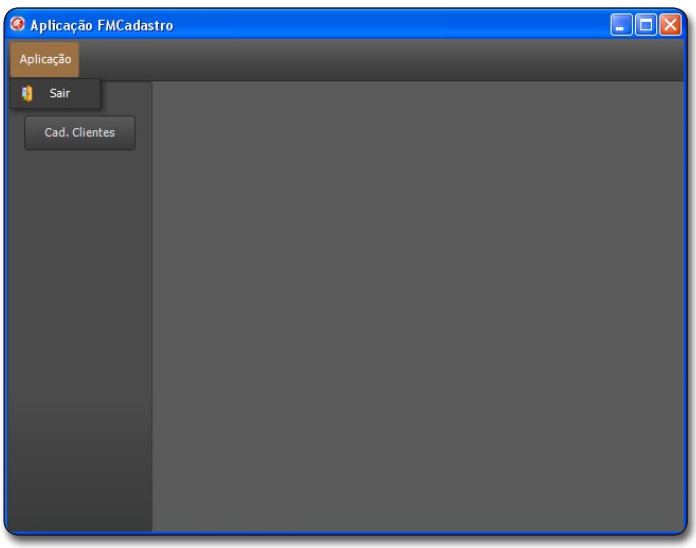


Figura 1. Estilo Visual da Aplicação com a utilização de um tema

Para criar uma animação 3D será inserido um objeto do tipo TViewPort3D. Através do TViewPort3D é possível exibir objetos 3D em um plano 2D e criar animações, utilizando um formulário padrão da FireMonkey, não necessitando que o mesmo seja do tipo TForm3D. O TForm3D também permite que sejam adicionados objetos e futuramente sejam executadas animações 3D. Neste exemplo o TViewPort3D foi optado pois mantém tudo em apenas um formulário, mas se o desenvolvedor optar por dividir o projeto em módulos, poderia ao invés de inserir os componentes no TViewPort3D, adicionarem um TForm3D da mesma maneira como é descrito neste artigo. Após adicionado o Viewport3D1, é interessante ajustar a propriedade Align do mesmo para alClient, fazendo com que ele ocupe todo o restante do formulário. Também é interessante mudar a cor de fundo do mesmo (propriedade Color) para #FF505050,que é a mesma cor de background utilizada pelo estilo Dark.

Para continuar é preciso adicionar um objeto do tipo TLayer3D no ViewPort3D1. O TLayer3D também é um objeto do tipo 3D utilizado para manter objetos 2D. A propriedade Projection do Layer3D1 deve ser alterada para pjScreen e a propriedade Align para alClient. A propriedade foi definida como pjScreen para que o objeto possa ser visto independente dos pontos da câmera, evidentemente, se

mantida a propriedade pjCamera, o objeto seria controlado pelos valores da propriedade Position. Também foi inserido um objeto do tipo TButton para exibir uma animação e ocultar o Layer3D1 quando o usuário clicar, alterando a propriedade Visible do mesmo para False. O código presente na **Listagem 1** demonstra como criar uma animação e posteriormente ocultar o Layer3D1.

Neste código é possível observar a grande utilidade do procedimento AnimateFloatWait(). Afinal, se o procedimento utilizado fosse o AnimateFloat(), provavelmente não daria tempo de ver a animação, uma vez que o próximo código é uma alteração da propriedade Visible para False do Layer3D1. Sendo assim, com a utilização do AnimateFloatWait(), o código de alteração da propriedade Visible só será executado após o término da animação. Também foi utilizado um Viewport3D1.Repaint para redesenhar o componente TViewPort3D.

Para criar a animação que será executada no momento em que o usuário clicar no botão de Cadastro de Clientes e exibir o Layer3D1, deverá ser inserido o código da **Listagem 2** no evento OnClick do botão mencionado anteriormente.

Listagem 1. Código do botão Fechar

```
procedure TFrPrincipal.btFecharClick(Sender: TObject);
begin
with Layer3D1 do
begin
    AnimateFloatWait('Position.Z', 5000, 1);
    Visible := False;
end;
  Viewport3D1.Repaint;
end;
```

Listagem 2. Código da Animação do Layer3D

```
procedure TFrPrincipal.btCadastroClick(Sender: TObject);
begin
with Layer3D1 do
  begin
    Position.Z := 500;
    Visible := True;
    AnimateFloat('RotationAngle.Y', 360, 1, TAnimationType.atInOut,
    TInterpolationType.itBack);
    Position.X := -300;
    AnimateFloatWait('Position.X', 250, 1);
    AnimateFloatWait('Position.Z', 0, 1);
end;
end;
```

Como pode ser observado neste código, o objeto Layer3D1 (que contém todos os outros componentes visuais) é utilizado para uma animação. Em primeiro momento é utilizada a propriedade Position.Z com valor de 500 para gerar um efeito de profundidade quando o mesmo for exibido. Vale a pena evidenciar que a propriedade Visible do Layer3D será definida como False em tempo de desenvolvimento futuramente para que a tela de cadastro de clientes seja exibida apenas na hora em que o usuário clicar no botão de cadastro, uma vez que tudo está em um mesmo formulário. Posteriormente um procedimento AnimateFloat() é utilizado para executar uma rotação de 360º no eixo Y, com duração de 1 segundo. Dando sequência a posição X é alterada para -300 para possibilitar a criação do efeito de deslizamento na horizontal do próximo procedimento AnimateFloat Wait(), lembrando que todos os procedimentos são executados em paralelo (AnimateFloat()) e ao chegar até este, a aplicação aguarda o término da animação para o efeito final que gera a aproximação do Layer3D que conterá os objetos para o cadastro.

Após criada a animação principal do Layer3D1 que será utilizada para sua exibição e os outros códigos descritos anteriormente, será criado um exemplo de conexão com o banco de dados utilizando o recurso de LiveBindings em conjunto com as ferramentas do dbExpress.

Para utilização do recurso de LiveBindings com oFireMonkey, foi utilizado o simples trecho de código SQL da **Listagem 3** em um banco de dados Firebird nomeado DADOS.FDB.

Após criado o banco de dados e uma simples tabela de clientes, foram adicionados ao formulário um objeto do tipo TSQLConnection para a conexão com o banco de dados Firebird, um do tipo TSimpleDataSet, para conter os dadosda tabela, e um TDataSource para o acesso, através das ferramentas dbExpress. No objeto SQLConnection1a propriedade ConnectionType foi definida como FBConnection e em Params, a propriedade database para C:\Sistema\Dados.fdb, bem como alterada para False a propriedade LoginPrompt, fazendo com que não seja exibida a tela de login. No objeto SimpleDataSet1, a propriedade connection foi definida como SQLConnection1 e em DataSet.CommandText inserido o código SELECT * FROM CLIENTES. Lembrando que para adicionar os campos da tabela clientes no SimpleDataSet1 é necessário dar um clique duplo sobre o mesmo e posteriormente um clique com o botão direito, para concluir selecionar a opção Add all fields. Para que os dados sejam atualizados no banco ao efetuar um Post, para gravação, e um Delete, para a exclusão, os eventos AfterDelete e AfterPost do SimpleDataSet1 deverão ter o código da **Listagem 4**.

Listagem 3. Código SQL da Tabela Clientes

```
CREATE TABLE CLIENTES(
  ID INTEGER NOT NULL,
NOME VARCHAR(50),
  ENDERECO VARCHAR(80),
  CONSTRAINT PKY_CLIENTES PRIMARY KEY(ID)
);
```

Listagem 4. Código de atualização dos Clientes

```
procedure TFrPrincipal.SimpleDataSet1AfterPost(DataSet: TDataSet);
begin
if SimpleDataSet1.ApplyUpdates(0) > 0 then
begin
    SimpleDataSet1.CancelUpdates;
    SimpleDataSet1.Refresh;
end;
end;
```

Através deste código os dados alterados da tabela clientes pela aplicação são atualizados no banco de dados. Após os processos descritos anteriormente, o DataSet1 também deve ser ligado ao SimpleDataSet1 através da propriedade DataSet.

Dando sequência, no Layer3D1 criado anteriormente para a exibição do cadastro dos clientes, foram criados 3 objetos do tipo TLabel para o ID, o NOME e o ENDERECO, e 3 do tipo TEdit para a exibição dos dados dos clientes e 1 TGroupBox para organizá-los.Depois de criados os TEdit, o próximo passo é efetuar o LiveBinding.

Com o Edit1 selecionado basta ir até a propriedade LiveBindings e selecionar a opção Link to DBField, posteriormente selecionar a opção ID. O próximo passo é executar o mesmo processo para o Edit2 e o Edit3, atribuindo os campos NOME e ENDERECO respectivamente. Concluído os processos acima, é necessário inserir um objeto do tipo TBindingNavigator e em sua propriedade BindScope, selecionar o objeto BindScopeDB1 criado automaticamente no momento em que foi efetuado o LiveBinding. Através do BindingNavigator1 inserido é possível navegar facilmente pelos registros da tabela.

No evento OnCreate do formulário o código SimpleDataSet1.Open é inserido para que o SimpleDataSet seja aberto no momento em que o formulário for criado, bem como no evento OnDestroy do mesmo o código SimpleDataSet1.Close para que o mesmo seja fechado.

Para concluir o projeto será adicionada uma animação do Tipo TBitmapAnimation. O objeto TBitmapAnimation descendente da classe TAnimation (FMX.Ani) é utilizado para animações com imagens. Ele deve ser ligado a propriedade TBitmap de um componente para que a imagem do mesmo possa ser alterada em determinado momento. Para a execução desta última etapa, no Layer3D1 é inserido um objeto do Tipo TImage. Com o objeto Image1 selecionado, para dar início a criação da animação, basta clicar na propriedade Bitmap do mesmo e selecionar Create New BitmapAnimation. Esta opção faz com seja criado um novo objeto do tipo TBitamapAnimation. Com o objeto BitmapAnimation1 selecionado, a propriedade StarValue deve ser clicada e a opção Edit deve ser selecionada para abrir a janela de seleção de um arquivo de Imagem. Lembrando que para carregar um arquivo de Imagem basta dar um clique no botão Load. Através de StarValue deste objeto é possível definir qual será a primeira imagem utilizada na animação, consequentemente, também a propriedade StopValue deve ser definida clicando em Edit e atribuindo a última Imagem da animação. Também deverá ser alterada a propriedade Trigger do BitmapAnimation1, onde a mesma será definida como IsMouseOver=true, ou seja, quando o mouse passar sobre a imagem, a animação irá iniciar a partir da imagem definida em StartValue e prosseguirá até a imagem definida em StopValue. Para o efeito inverso, quando o Image1 perder o foco do mouse, será utilizada a propriedade TriggerInverse como IsMouseOver=false. A propriedade TriggerInverse faz que a animação comece da imagem StopValue e retorne até a imagem definida em StartValue quando o foco do mouse for perdido neste caso. Para finalizar, na propriedade Bitmap do Image1, deve ser selecionada a opção Edit para definir a imagem padrão de exibição do mesmo quando não for executado o objeto BitmapAnimation1. A **Figura 2** demonstra a aparência final da Aplicação.

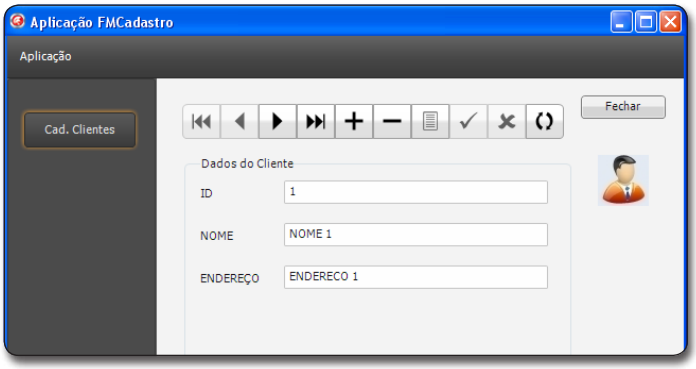


Figura 2. Estilo Visual final da Apliação com o cadastro de clientes exemplo

Conclusão

OFireMonkey é uma plataforma de ponta completa para a criação de uma aplicação através do Delphi. Oferece vários recursos, dentre eles efeitos gráficos, conexão com bancos de dados, performance e portabilidade, não necessitando de componentes adicionais. Ela possui um futuro promissor em virtude da sua portabilidade que poderá ser estendida para outras plataformas, bem como pelo fato de oferecer ao desenvolvedor uma nova possibilidade não muito presente em versões anteriores do Delphi que visa a utilização de recursos visuais, fazendo que o desenvolvedor não tenha que recorrer a outras ferramentas quando desejar obter um visual mais refinado para a sua aplicação. Como foi possível observar, a utilização de recursos de animação do FMX é bem simples, uma vez que já existem procedimentos praticamente prontos para criar vários efeitos presentes em diversos componentes, como é o caso do AnimateFloat(), lembrando também que as animações podem ser executadas em paralelo ou em série de acordo com o desejo do desenvolvedor.

Para concluir, o FireMonkey dá vários indícios de que é uma plataforma que veio para ficar e facilitar a vida do desenvolvedor que gosta e prefere utilizar o Delphi, embora evidentemente tenha que se atualizar um pouco e se adaptar a algumas mudanças como toda nova tecnologia prevê.

Giuliano Scombatti Pinto
giuliano@sygnux.com.br
É Analista Desenvolvedor da Sygnux Software (www.sygnux.com.br) localizada em Monte Alto/SP, atuando também com desenvolvimento Web e Android. Formado pela Fatec/TQ, começou a desenvolver com Delphi em 2003. Atualmente trabalha com Delphi, Java, PHP e Flash/Flex. Professor de Informática de Projetos Sociais da Prefeitura Municipal de Monte Alto/SP.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

DevMedia

DÊ UM SALTO EM CONHECIMENTO!

Acesse o maior portal para desenvolvedores da América Latina!

17 mil posts

280 mil cadastrados

5 milhões de page-views por mês

http://www.devmedia.com.br/

DevMedia

Aplicações Windows
Introdução ao desenvolvimento de apli

Resumo DevMan

De que trata o artigo:

O lançamento XE2 dos produtos Embarcadero trouxe novidades no Delphi, PHP e também no Prism. Neste artigo é abordada uma das novidades do Prism XE2, a possibilidade do desenvolvimento de aplicações voltadas para o Windows Phone. Dessa forma, a premissa principal do artigo é dar um start neste desenvolvimento.

Em que situação o tema é útil:

Desenvolvimento de aplicativos para smartphone utilizando Prism XE2. Até tempos atrás, o que se via na indústria de software era sua predileção e foco voltado para a construção e posterior comercialização de aplicativos para as já tradicionais plataformas Desktop e Web. No entanto, em tempos mais recentes o que se viu foi esse direcionamento sendo voltado também para uma nova gama de aplicativos, agora destinados a um nicho totalmente emergente, que é o das plataformas móveis. Neste quesito, especificamente para o mundo .NET, o foco do momento é o desenvolvimento de aplicativos para o mais recente sistema operacional móvel da Microsoft, o Windows Phone 7, que substitui o anterior Windows Mobile. Como já dito, através do Prism, os desenvolvedores Delphi são capazes de usufruir da gama de possibilidades .NET, inclusive a construção de aplicativos deste tipo.

Aplicações Windows Phone no Prism XE2:

O Delphi Prism se popularizou no meio Delphi por ser uma nova opção no que diz respeito ao desenvolvimento de aplicações. O seu diferencial se dá pelo fato de que, por se tratar de um plug-in, se integra devidamente ao principal IDE para o desenvolvimento baseado no Microsoft .NET Framework, o Visual Studio. Assim sendo, possibilita que o desenvolvedor Delphi faça uso dos mais variados recursos deste framework para a construção de aplicações desta natureza, utilizando para isso uma linguagem muito próxima do Pascal (utilizada no Delphi tradicional), conhecida como Oxygene. Em meio a novos recursos e novidades da versão XE2, o nome do produto foi alterado sendo agora chamado apenas de Prism XE2. Neste meio tempo o mais recente sistema operacional móvel da Microsoft, o Windows Phone 7, se estabelecia no mercado fazendo o desenvolvimento de aplicativos para esta plataforma ganhar certo destaque. O Visual Studio eficazmente proporciona templates de projetos para a construção destas aplicações, que utilizam uma versão especial do Microsoft Silverlight. Logo, através do Prism pode-se fazer proveito de tais recursos do IDE para a construção de aplicações para Windows Phone. Desta forma, o artigo visa apresentar todo um embasamento teórico que envolve este tipo de desenvolvimento, assim como demonstrar de maneira prática os recursos disponíveis pelo IDE para tal.

Nos dias atuais o desenvolvimento de aplicativos móveis vem numa crescente na mesma medida em que seus dispositivos se estabelecem num mercado que não para de evoluir. Estes dispositivos são conhecidos como smartphones, possuem funcionalidades tão avançadas quanto o de um computador tradicional, e que são executadas por aplicativos através de seu sistema operacional. Este cenário atual de smartphones se contrasta à sua geração anterior formada essencialmente por telefones celulares simples, que possuíam apenas os recursos

Phone no Prism XE2
cativos para esta plataforma móvel

Fabrício Hissao Kawata

básicos como chamadas de voz, mensagens de texto e adicionalmente alguns jogos mais triviais.

Dentre os SOs existentes atualmente alguns já se estabelecem como dominantes, podendo-se destacar o iOS, também conhecido como iPhone OS, Android e Windows Phone 7. Dessa forma, no que tange o desenvolvimento móvel a escolha por uma determinada plataforma é o primeiro passo a ser dado. Especificamente para o ambiente .NET, obviamente que a opção que se encaixa no contexto é o Windows Phone 7, que é a nova aposta da Microsoft para este mais recente nicho de mercado. Esta plataforma, aliás, veio para substituir sua antecessora, a do Windows Mobile.

A plataforma Windows Mobile surgiu em meados de 2000 e rodava exclusivamente sobre um hardware denominado Pocket PC. Em vista do sistema operacional Windows, aplicativos já conhecidos do ambiente Desktop ganharam sua versão móvel, como Word, Excel, Internet Explorer, entre outros. Além disso, foi através do Windows Mobile que a Microsoft proporcionou aos desenvolvedores a primeira oportunidade para criação de aplicativos voltados para dispositivos móveis. Isto porque além do próprio sistema operacional móvel, ela lançou um conjunto de ferramentas de desenvolvimento para suas aplicações nativas.

Na visão do desenvolvedor, o que marcou essa geração Windows Mobile foram alguns problemas pontuais que afetavam direta ou indiretamente a questão mercadológica. Em outras palavras, se o desenvolvedor da época tinha a intenção de produzir aplicações comercialmente, encontrava certas barreiras naturais. Uma delas era o público a ser atingido, uma vez que os aparelhos eram caros para o padrão popular, a ponto da plataforma se voltar para os usuários corporativos em detrimento à grande massa. Outro fator diz respeito à navegação Web, em vista da baixa velocidade, do tamanho inadequado da tela dos dispositivos e dos altos preços dos planos de dados. Por fim, o desenvolvedor ainda tinha o entrevero da portabilidade, em que a cada nova versão lançada do sistema operacional, novos e distintos dispositivos eram lançados, cada qual com seus próprios recursos e sua resolução de tela, suficientes ou não para atender suas aplicações atuais. Assim sendo, o desenvolvimento ficava fadado a testes na maior variedade de dispositivos possíveis, causando transtornos não medidos.

Windows Phone 7

Em vista dos problemas já conhecidos e reportados pelos consumidores (usuários e desenvolvedores), e também muito em função também do sucesso recente de outras plataformas como iPhone e Android, a Microsoft resolveu então fazer um reboot de sua plataforma móvel. Eis que surge então o Windows Phone 7 Series, com a premissa de sanar os "problemas" da plataforma anterior com medidas como, por exemplo, definir um conjunto padronizado de hardware, além de agregar recursos pertinentes ao cenário atual, como o suporte a multitouch (multi-toque) e melhorar a experiência com o usuário. Mais recentemente a plataforma foi então renomeada, tendo seu nome encurtado para simplesmente Windows Phone 7 (WP7).

Visando os desenvolvedores, a Microsoft oferece também um SDK exclusivo, e faz uso de duas tecnologias já consagradas em meio a sua comunidade de desenvolvedores: o .NET Framework e o Silverlight. Além disso, o desenvolvimento se dá no IDE do Visual Studio. Dessa forma ela garante que haja um reaproveitamento de conhecimento, diminuindo a curva de aprendizagem e facilitando a adoção deste tipo de desenvolvimento por parte dos profissionais que já trabalham com as tecnologias citadas. Ainda sem deixar de mencionar, dentre as ferramentas de produção é disponibilizado também um eficiente emulador que auxilia e muito na realização de testes, em vista da não necessidade de um aparelho físico à disposição para o desenvolvimento.

Com o iminente lançamento de aplicações para a plataforma, a Microsoft tratou também de anunciar a criação de sua própria loja de aplicativos. Dessa forma, proporciona ao profissional um local centralizado para a disponibilização das aplicações, seja de forma comercial ou não. Para que o desenvolvedor possa participar da loja e assim ter o direito de expor nela seus aplicativos, ele deve aderir ao também anunciado programa de desenvolvimento denominado Windows Phone 7 Developer Program.

Nota do DevMan

Software Development Kit, ou simplesmente SDK, são pacotes contendo recursos e ferramentas de desenvolvimento que visam auxiliar no trabalho do programador promovendo uma melhor integração com determinado software. Geralmente estes pacotes são disponibilizados pelos proprietários ou mentores do referido software como, por exemplo, o SDK do Windows Phone 7 provido pela Microsoft.

Especificações

Conforme foi dito, com o WP7 uma das maiores pretensões da Microsoft era evitar grande parte dos problemas conhecidos de sua plataforma móvel anterior. Assim sendo, uma das primeiras medidas tomadas foi estipular uma lista de requisitos mínimos que um dispositivo que irá rodar seu sistema operacional

deverá possuir. Com esta medida, há uma limitação natural do conjunto de configurações de hardware disponíveis, todavia, garante-se uma padronização no que diz respeito à execução dos aplicativos para a plataforma, bem como a experiência do usuário. Olhando pelo lado do consumidor, isso quer dizer que um usuário deverá ser capaz de aproveitar o seu conhecimento sobre a utilização de um dispositivo Windows Phone, seja qual for o aparelho. Pela visão do desenvolvedor, fica garantido que o desenvolvimento de um aplicativo dentro das especificações de hardware propostas proporcionará uma portabilidade eficiente, uma vez que sua execução se dará da mesma forma em aparelhos distintos.

A lista de requisitos mínimos do hardware envolve basicamente os seguintes itens:

- Câmera de 5 megapixels (mínimo), com flash, além de um botão de câmera;
- Memória RAM de 256MB (mínimo);
- Memória Flash de 8GB (mínimo);
- Unidade gráfica (GPU) com suporte a DirectX 9;
- Processador ARMv7 Cortex/Scorpion (mínimo);
- GPS;
- Sensores: acelerômetro, bússola, luz e proximidade;
- Tela com capacidade multitouch, com 4 ou mais pontos de contato;
- Resolução de tela 800 x 400.

Além dos itens listados, outro requisito essencial nos aparelhos WP7 ganha destaque pela sua nítida notoriedade, porá exigência de três botões frontais no hardware. Conforme pode ser visto na **Figura 1**, os botões em questão são os tradicionais Back (Voltar), Start (Iniciar) e Search (Pesquisar) da plataforma Windows. Por este motivo e também em vista da localização, eles se tornam um importante item de marketing visual, já que apenas pela visualização do aparelho é possível certificar que se trata de um aparelho Windows Phone.

Figura 1. Três botões frontais do hardware

Fazendo uma análise prévia com base na lista de requisitos mínimos, fica fácil concluir que um smartphone com Windows Phone 7 será essencialmente poderoso e com recursos suficientes para rodar os mais variados tipos de aplicações que venham a ser desenvolvidas.

Arquitetura

Em termos de arquitetura, a plataforma do Windows Phone 7 é um ponto a favor dos desenvolvedores, facilitando o desenvolvimento de aplicações ricas. Isto é justificado pelos recursos providos pelos quatro componentes principais da arquitetura: componentes de Runtime (Runtime components), Tools (ferramentas), serviços de Cloud e Portal (Cloud services e Portal services).

Runtime Components

Estes são os componentes de runtime, ou seja, de tempo de execução. Refletem diretamente no processo de desenvolvimento para a plataforma Windows Phone 7. Neste quesito dois são os caminhos principais a serem seguidos, que consistem na construção de aplicações ricas com a tecnologia Silverlight, além de jogos com o XNA.

Em relação ao Silverlight, o desenvolvimento para a plataforma móvel é muito próximo ao seu desenvolvimento tradicional para Web, uma vez que, a maioria das funcionalidades de sua versão padrão também está disponível em sua versão exclusiva para WP7. Para quem não se lembra, o Silverlight inclusive já foi abordado em artigos anteriores nesta revista. Já o XNA é o framework da Microsoft voltado para o desenvolvimento de games para PC's Windows, consoles Xbox 360 e agora mais recentemente dispositivos WP7.

Os componentes de Runtime incluem ainda um tipo de elemento essencial à geração atual de smartphones, que é representado pelos elementos de mídia (media elements). Estes elementos são muito próximos dos utilizados pelo Zune, o dispositivo multimídia da Microsoft, e permitem que você manipule os controles de áudio e vídeo de uma maneira customizada por sua aplicação. Ainda falando sobre itens essenciais ao smartphone, a parte de chamadas telefônicas também se encaixa neste grupo de componentes. Isso quer dizer que é possível fazer uso das funções do telefone em suas próprias aplicações. Além disso, fica a cargo do próprio WP7 garantir que elas se comportem de maneira adequada quando os eventos inerentes ao telefone ocorrerem como, por exemplo, ao receber uma simples chamada telefônica no momento da execução da aplicação. De maneira oposta, a Microsoft também estabelece como regra que as aplicações de forma alguma podem interferir nos recursos básicos do telefone, como chamadas e mensagens SMS. Em vista disso, a plataforma do Windows Phone 7 provê ainda uma série de manipuladores de eventos relacionados ao ciclo de vida da aplicação, permitindo que seus dados sejam salvos (na memória ou em outro dispositivo de armazenamento). Num caso real, após uma interrupção da execução da aplicação em função do acionamento de um recurso do aparelho, o aplicativo então poderia ser facilmente restaurado exatamente no ponto onde parou.

Ainda falando especificamente sobre armazenamento de dados, o WP7 utiliza o conceito de Isolated Storage do Silverlight, que é um mecanismo de armazenamento isolado e seguro, que se utiliza de maneiras padronizadas para associar código com os dados salvos. Por fim, os sensores também são outros itens presentes nos componentes de Runtime, incluindo aí recursos como bússola, GPS e acelerômetro que os aplicativos desenvolvidos para a plataforma poderão fazer uso.

Cloud Services

Juntamente com o lançamento do Windows Phone 7, a Microsoft fornece aos desenvolvedores uma série de serviços na nuvem (cloud) para serem utilizados no desenvolvimento de aplicativos. Dentre estes, um dos mais utilizados é o serviço de notificação (notification service). Através dele, você é capaz de "empurrar" dados e mensagens para o dispositivo WP7, sem que a aplicação tenha que necessariamente fazer requisições para obter estas informações. Com o detalhe de tudo isso estar disponível de forma simples, não envolvendo operações complexas. Este tipo de recurso é muito empregado em alertas sistêmicos, informações metereológicas e coisas do tipo.

Outro cloud service muito utilizado no WP7 é o serviço de localização que, como o próprio nome já sugere, possibilita à aplicação acessar as informações sobre a localização física do usuário. Para tal, há duas formas básicas: consultar manualmente uma localização através do uso de uma API ou fazer uso de eventos específicos que notificam a aplicação quando a localização do usuário é alterada. Adicionalmente, os níveis de precisão e as informações de localização ainda podem ser obtidos através do GPS do hardware e da torre de celular da própria operadora usada.

A Microsoft disponibiliza também uma API voltada aos serviços da Xbox Live e Windows Azure. Dessa forma, as aplicações podem facilmente acessar o profile e releases de uma conta da Xbox Live ou ainda se utilizar dos recursos e serviços Web da plataforma do Windows Azure. Por fim, essa parte de serviços na nuvem do Windows Phone 7 provê ainda a possibilidade de se trabalhar com a plataforma SQL Azure, que é o banco de dados relacional na nuvem da Microsoft, além de um novo controle para se desenvolver e prover às aplicações as funcionalidades do Bing Maps (o sistema de mapas da Microsoft).

Portal Services

Da mesma forma que os diversos serviços na nuvem, a Microsoft também disponibiliza outro tipo de serviço denominado serviços de portal, ou Portal Services. Eles estão relacionados diretamente à implantação (deploy) e distribuição dos aplicativos desenvolvidos. Isto porque os desenvolvedores invariavelmente precisam de um registro junto à Microsoft para que possma distribuir suas aplicações construídas para o Windows Phone 7. Por conseguinte, essa distribuição é feita através de um site específico, denominado App Hub (ver endereço na seção Links). Neste mesmo local o desenvolvedor poderá se registrar ao programa, além de obter todas as ferramentas necessárias ao desenvolvimento. Além disso, através do App Hub o desenvolvedor pode ainda gerenciar e distribuir atualizações dos seus aplicativos, estabelecendo se elas serão gratuitas ou terão custo. Outro site relacionado é o Windows Phone Marketplace, que é o local onde os usuários poderão então baixar, de forma gratuita ou através de compra, estas aplicações distribuídas.

Ainda neste grupo de serviços temos a sua validação e certificação. Eles servem essencialmente para garantir que nenhum código malicioso esteja presente nas aplicações e consequentemente no dispositivo WP7. Sendo assim, estão envolvidos já no momento do desenvolvimento, ao se utilizar o próprio emulador de dispositivo. Os serviços de validação e certificação então garantem a ausência de qualquer coisa que possa vir a se tornar uma potencial ameaça ao telefone (por exemplo, fazendo-o travar durante a execução do aplicativo) ou que viole os termos de acordo de serviço do mesmo.

Outro serviço relacionado ao App Hub a ser citado é o de faturamento (billing service), uma vez que a própria plataforma WP7 provê funcionalidades para a criação de aplicações de caráter gratuito (free), pago (paid) ou de experimentação (trial). Isto é obtido através de funcionalidades que restringem determinadas funções do aplicativo, dependendo da situação do usuário. Um exemplo clássico é o das aplicações que possuem uma versão de avaliação temporária, que possui certa quantidade limitada de recursos. Ao fazer a compra do aplicativo, o usuário então recebe uma versão completa da mesma, com todas suas funcionalidades habilitadas.

Tools

Fechando a arquitetura do Windows Phone 7 temos as ferramentas de desenvolvimento (developer tools). Neste ponto a Microsoft primou por manter seu IDE líder de mercado, o Visual Studio, como sendo o ambiente de desenvolvimento principal. Justamente por isso, os desenvolvedores Delphi podem facilmente desfrutar do desenvolvimento nativo para Windows Phone 7 em razão da integração do Prism XE2 com o VS. Outro ponto prezado foi prover à ferramenta Expression Blend as capacidades necessárias para a construção de designs voltados para aplicações da plataforma WP7, possibilitando que desenvolvedores e designers se co-relacionem mais facilmente. O melhor de tudo ainda é que foram disponibilizadas versões gratuitas dessas ferramentas (Visual Studio e Expression Blend), voltadas especificamente para o desenvolvimento móvel. No caso do IDE, apesar de gratuito, ele possui uma licença comercial que garante que você possa comercializar as aplicações desenvolvidas por você.

Silverlight

O Silverlight é uma tecnologia que provê o desenvolvimento de aplicações ricas para internet, apoiada pelo framework .NET e suas linguagens relacionadas, tais como Oxygene, através do Prism XE2. Tais aplicações são popularmente conhecidas como RIA, um acrônimo para Rich Internet Applications, que quebram o paradigma das aplicações Web tradicionais e seguem o padrão da Web 2.0. Exemplificando, no modelo antigo, uma simples ação numa página provocava uma atualização da página inteira. Nesta nova proposta, apenas porções da página são atualizadas, conforme a necessidade, resultando numa agilidade tão eficiente quanto numa aplicação Desktop, aumentando a experiência do usuário.

O runtime do Silverlight é sintetizado em um simples plug-in para o browser, isto porque todo o código que define a interface de usuário da aplicação é executado no cliente. O código utilizado para isso é definido em uma linguagem declarativa denominada XAML (eXtensible Application Markup Language), baseada em XML. Graças a ela é possível se construir a interface da aplicação

totalmente isolada do código de negócio. É neste ponto que entra em cena o Expression Blend, que provê exclusivamente a criação de interfaces da aplicação, por um profissional designer e consequentemente o aproveitamento de todo o código XAML produzido, complementado com o código de programação no próprio Visual Studio. A forma declarativa da linguagem XAML lembra em certos pontos a do HTML. Assim sendo, a declaração de um simples botão poderia se dar da seguinte forma:

```
<Button Width="50" Height="50">Clique aqui</Button>
```

Fazendo menção exclusivamente à plataforma do Windows Phone 7, o Silverlight provê ao desenvolvedor uma robusta coleção de controles para serem utilizados nas mais diversas aplicações para os mais distintos fins (Ex: entrada de dados do usuário, exibição de dados e botões de ação). A grande maioria deles já vem personalizada para o novo estilo de interface de usuário adotada no WP7, denominada Metro.

Em termos práticos, um desenvolvedor Silverlight que já utilizava a tecnologia para a construção de aplicações para Web, está plenamente apto a desenvolver para Windows Phone 7. Isto porque, por se tratar essencialmente da mesma tecnologia, não se perde o conhecimento adquirido. É necessário somente tomar conhecimento sobre as nuances relativas às versões do Silverlight (tradicional e móvel) como, por exemplo, os controles que são ou não suportados para o telefone. Em muitos casos, uma aplicação Silverlight "tradicional" pode ser eficazmente portada para Windows Phone.

Preparando o ambiente de desenvolvimento

Uma vez apresentadas as ferramentas de desenvolvimento envolvidas na arquitetura da plataforma Windows Phone 7, tomamos o conhecimento superficial do que é necessário para se iniciar a construção de aplicativos móveis: Visual Studio 2010, XNA Game Studio, Expression Blend for Windows Phone 7 e Embarcadero Prism XE2. Os três primeiros itens estão presentes no Windows Phone SDK (ver seção Links), que se encontrava na versão 7.1 durante a escrita deste artigo. Adicionalmente ainda são instaladas as ferramentas Silverlight necessárias bem como o já mencionado emulador que será utilizado para se testar as aplicações construídas na própria máquina local. Já o Prism pode ser obtido diretamente do site da fabricante e, no caso do desenvolvedor Delphi, deverá ser o primeiro a ser instalado no ambiente. Dessa forma, será instalado um Shell do Visual Studio 2010 já com o plug-in do Prism XE2 integrado, sendo complementado posteriormente pelos recursos do SDK do Windows Phone.

Figura 2. Alguns tipos de projeto para Windows Phone

Prática - construindo a primeira aplicação

Quando se está aprendendo uma linguagem ou tecnologia nova, corriqueiramente é utilizado como primeiro exemplo prático as já tradicionais aplicações "Hello World". Seguindo esta linha, a primeira aplicação a ser construída para a plataforma Windows Phone 7 utilizando o Prism XE2 será um simples "Alô DevMedia". Dessa forma, já num primeiro momento será possível ver alguns dos tipos de projetos disponíveis para este tipo de desenvolvimento, a forma de codificação a ser utilizada, bem como a aplicação teste sendo executada no emulador. Somente como uma prévia, a **Figura 2** exibe os templates de projetos disponíveis no IDE para a construção de aplicações tradicionais WP7, desde uma simples Windows Phone Application até uma Class Library. Itens estes que ficam localizadas no sub-grupo Silverlight for Windows Phone, do grupo Oxygene for .NET.

Cada template deste ao ser selecionado, monta automaticamente toda a infraestrutura básica requerida para o referido tipo de projeto. Dito isto, para este exemplo, a preferência será pela construção de uma Windows Phone Application que, quando carregada no IDE, já exibe em

sua parte principal o emulador contendo uma página simples pré-definida, bem como o código XAML responsável por esta definição. Esta primeira impressão é mostrada na **Figura 3**.

Em termos de estrutura, a que foi criada pelo template de projeto pode ser conferida na Solution Explorer do Visual Studio. Notoriamente percebe-se que foram criados vários arquivos que podem, basicamente, ser divididos em dois grupos: arquivos de imagem (.png e .jpg) e arquivos XAML. Em relação ao primeiro grupo não cabe nenhum comentário adicional neste momento, visto que se refere somente a aspectos visuais da aplicação e serão discutidos posteriormente. Já os dois arquivos XAML exercem funções específicas dentro do contexto do projeto. App.xaml é figura presente em todos os projetos de aplicações para Windows Phone 7, pois representa a aplicação em si, onde estão declarados os recursos globais que estarão disponíveis para o aplicativo como um todo. Na **Listagem 1** é apresentado um trecho do código padrão presente neste arquivo, onde nota-se a presença de uma seção sugestivamente denominada Application Resources. Nela estarão os recursos da aplicação, além de outra seção relacionada ao ciclo de vida da aplicação. Nesta última, nomeada como Application LifetimeObjects, é declarado um objeto PhoneApplicationService que é um serviço global utilizado para se capturar os principais eventos relacionados ao telefone em si. Sendo assim, para cada uma de suas propriedades são associados os manipuladores de tais eventos do telefone.

Conforme pode ser visto, no total são quatro os eventos que controlam o ciclo de vida da aplicação:

• Application_Launching: o código deste evento será executado quando a aplicação efetivamente for disparada;

• Application_Closing: o código deste evento ocorrerá no fechamento de uma aplicação, que pode ser conseguido através do botão Back (voltar);

• Application_Activated: o código deste evento será executado no momento em que a aplicação é ativada, ou seja, quando ela é trazida para primeiro plano;

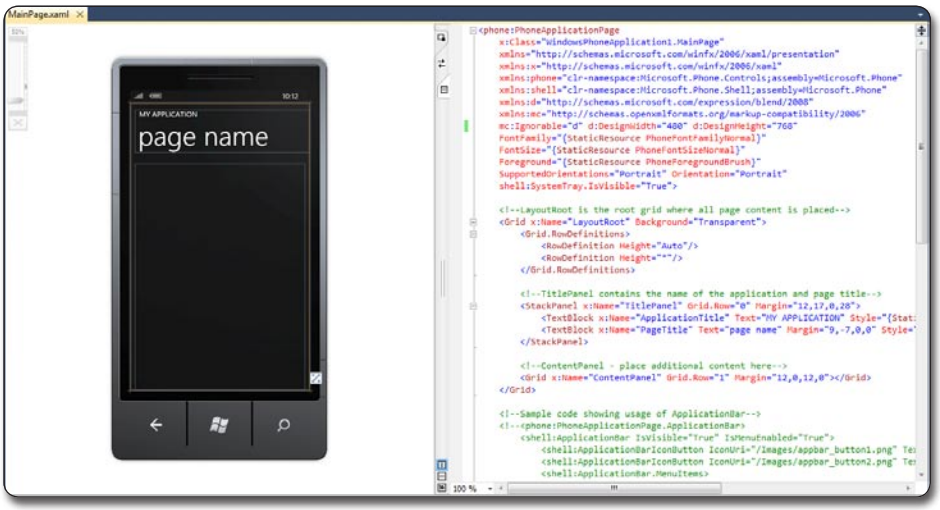


Figura 3. Visão panorâmica do IDE de um projeto Windows Phone Application carregado

Listagem 1. Trecho do código presente em App.xaml

```
<Application.Resources>
</Application.Resources>

<Application.ApplicationLifetimeObjects>
  <shell:PhoneApplicationService
    Launching="Application_Launching" Closing="Application_Closing"
    Activated="Application_Activated" Deactivated="Application_Deactivated"/>
</Application.ApplicationLifetimeObjects>
```

• Application_Deactivated: opostamente ao evento anterior, o código deste será executado quando a aplicação for desativada (quando sai do primeiro plano).

O detalhe aqui é que os eventos citados recebem como valor o nome de um método como, por exemplo, Application_Closing. Este método, assim como os demais, fica localizado no arquivo de code-behind (App.xaml.cs) do próprio XAML. Nele podemos verificar todo o código de tais métodos e a forma padrão apresentada por um projeto Windows Phone 7, de criar os manipuladores de eventos totalmente vazios. Dessa forma, cabe ao desenvolvedor adicionar alguma lógica customizada nestes eventos, caso necessite. São nos arquivos de code-behind que você usará efetivamente a linguagem Oxygene para definir o código de negócio envolvido necessário.

O outro arquivo XAML presente é o MainPage.xaml, que representa a página principal da aplicação. No entanto, seria perfeitamente possível ter outro arquivo no lugar deste, com outra nomenclatura, mas o importante aqui é frisar que esta página já é pré-criada pelo IDE como forma de agilizar o desenvolvimento. Dessa forma, o que vale ser mencionado é o teor do seu conteúdo, que define o aspecto inicial da aplicação, ou seja, o XAML declarado é o que define a interface da aplicação, que pode ser vista neste primeiro momento no emulador. Na **Listagem 2** é exibido um trecho do código desta janela principal, onde basicamente podemos ver nitidamente sua divisão em dois blocos principais: a declaração inicial, referente às diretivas e propriedades da página, e a seção da tag <Grid>, voltada ao conteúdo.

Um detalhe importante a ser mencionado sobre a estrutura básica de uma página XAML de uma aplicação Window Phone, é com relação ao seu objeto PhoneApplicationPage declarado no início do código. Este elemento é o objeto pai da aplicação, ou seja, é o que estará no topo da hierarquia. Eventuais telas adicionais serão consideradas filhas deste objeto. Sendo assim, todas as aplicações Windows Phone

requerem uma instância deste controle. Já a presença do objeto Grid é referente ao layout da interface de usuário da aplicação, e servirá de container para o conteúdo da página. Em outras palavras, ele será o elemento raiz, e qualquer objeto ou informação que se deseje exibir deverá ser colocado dentro dele.

A página MainPage também possui seu arquivo de code-behind associado, que é onde ficará localizado todo o código responsável por interagir com os controles em tela, bem como seus manipuladores de eventos. A **Listagem 3** mostra o construtor da classe MainPage que é definido neste arquivo e a proximidade da linguagem com a utilizada no Delphi tradicional.

Listagem 2. Trecho do código presente em MainPage.xaml

```
<phone:PhoneApplicationPage
  x:Class="WindowsPhoneApplication1.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=
  Microsoft.Phone"
  xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
  FontFamily="{StaticResource PhoneFontFamilyNormal}"
  FontSize="{StaticResource PhoneFontSizeNormal}"
  Foreground="{StaticResource PhoneForegroundBrush}"
  SupportedOrientations="Portrait" Orientation="Portrait"
  shell:SystemTray.IsVisible="True">

 <Grid x:Name="LayoutRoot" Background="Transparent">
  ...
 </Grid>

</phone:PhoneApplicationPage>
```

Listagem 3. Declaração do construtor da classe MainPage

```
type
 MainPage = public partial class(PhoneApplicationPage)

 public
  constructor ;
 end;

implementation

constructor MainPage;
begin
 InitializeComponent()
end;
```

Arquivos complementares

Além dos arquivos principais criados junto ao projeto, que foram explicitados anteriormente, pudemos comprovar através da Solution Explorer a presença de outros arquivos, tais como os de imagem (.png e .jpg). Apesar de serem poucos mencionados em outras literaturas, estes arquivos acabam tomando certa importância ao se desenvolver uma aplicação para fins comerciais. Isto porque, nestes casos os detalhes visuais servem como atrativo para se ganhar o apreço do usuário. Por exemplo, o arquivo Splash ScreenImage.jpg, que mostra um relógio (**Figura 4**), é a imagem que será exibida quando o aplicativo está sendo carregado. Em outras palavras, será o primeiro impacto visual que o usuário terá ao iniciar sua aplicação. De forma similar, o arquivo ApplicationIcon.png refere-se ao ícone do aplicativo, que é o que será visto na lista de aplicativos instalados no aparelho. Assim sendo, você poderá customizar estes recursos, substituindo estes itens por outras imagens de sua preferência. O importante, neste caso, é manter o nome padrão destes arquivos, visto que são os nomes utilizados nas configurações de runtime do projeto.

Figura 4. Imagem SplashScreenImage na tela do dispositivo WP7

Personalizando o conteúdo

Agora que foi exposto uma prévia sobre a estrutura do código presente no arquivo MainPage.xaml, podemos facilmente dar continuidade à proposta do exemplo e assim personalizar os dizeres do texto principal exibido na aplicação. Conforme visto, o conteúdo necessário deverá ser colocado dentro do objeto Grid. Dessa forma, por padrão, o projeto já insere neste trecho outro container contendo controles Silverlight do tipo TextBlock, utilizados para apresentação de textos estáticos. Na verdade são adicionados dois, contendo o nome da aplicação e o título da página, conforme o trecho exibido na **Listagem 4**.

Listagem 4. Trecho de código que define o conteúdo inicial da página principal

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
   <TextBlock x:Name="ApplicationTitle"
         Text="MY APPLICATION"
         Style="{StaticResource PhoneTextNormalStyle}"/>
   <TextBlock x:Name="PageTitle"
         Text="page name"
         Margin="9,-7,0,0"
         Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

Os textos definidos como sendo o nome da aplicação e o título da página neste momento, estão valorados como "MY APPLICATION" e "page name" respectivamente. Isso se dá através da propriedade Text dos objetos TextBlock. Sendo assim, para

personalizar tais dizeres de acordo com a proposta do exemplo, bastaria atribuir um novo valor às referidas propriedades. A **Figura 5** mostra o emulador em execução com a interface inicial da aplicação já personalizada com os dizeres "Minha Aplicação WP7" e "Alô DevMedia". Vale ressaltar que ao rodar a aplicação, o próprio Visual Studio inicia o emulador de um aparelho Windows Phone 7.

Figura 5. Aplicação de exemplo em execução

Aproveitando a menção feita sobre o emulador, ele também provê ao desenvolvedor a realização da mudança de orientação do dispositivo, ou seja, ir da posição vertical para a horizontal. Isto é possível através de um recurso de orientation presente em sua toolbar. Obviamente, este tipo de mudança pode afetar a interface de usuário da aplicação, sendo assim, a própria API do Windows Phone já provê manipuladores de eventos para estes casos de mudança de orientação. Em pontos como este é que se confirma a importância de um emulador, pois dessa forma, através dele é possível se verificar de que forma a aplicação irá se comportar em um dispositivo real.

Conclusão

Anteriormente, nesta própria revista, já foram abordados temas relacionados ao Delphi Prism envolvendo aplicações Web, como Silverlight e ASP.NET, e Desktop por meio do Windows Presentation Foundation, o já popular WPF. Com este artigo vimos mais uma opção de desenvolvimento com o Prism através de sua recente versão XE2. O tipo de aplicação abordado aqui atinge um novo nicho de mercado emergente, que é o de dispositivos móveis. Neste quesito, atualmente os destaques ficam por conta dos aparelhos com sistemas operacionais iOS, Android ou Windows Phone 7. Sobre este último, conforme já foi dito, a ferramenta ideal para desenvolvimento de suas aplicações cliente denomina-se Visual Studio (Prism), que provê recursos e funcionalidades suficientes para atender os mais variados tipos de aplicativos para a plataforma WP7. Em função disto, o artigo tentou ser o mais abrangente possível, visando dar um start no desenvolvimento destes aplicativos e assim servir de referência para outros futuros mais específicos.

Fica como dica, principalmente para aqueles que têm interesse no desenvolvimento para plataformas móveis no meio Delphi, além do tema abordado pelo presente artigo, explorar outros novos recursos apresentados com a família XE2. Só para exemplificar, no Delphi X2 temos os novos DataSnap Mobile Connectors, enquanto que no RadPHP XE2 é possível produzir aplicativos Web para dispositivos móveis Android e iOS.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 5 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas além de participar do programa Beta Tester Delphi Prism, da Embarcadero.
Blog: prismsilver.blogspot.com

Embarcadero Prism XE2 Enterprise - Trial
https://downloads.embarcadero.com/free/prism

Microsoft App Hub
http://create.msdn.com/en-US/

Windows Phone SDK 7.1
http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=27570

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Desenvolvimento OO
Persistindo objetos com DORM

Resumo DevMan

De que trata o artigo:

Da aplicação do desenvolvimento orientado a objetos, seus pilares e boas práticas, utilizando recursos nativos do Delphi XE 2.

Em que situação o tema é útil:

Entender a orientação a objetos como um todo, entender o que o Delphi XE 2 oferece sobre o assunto e como aplicar em um projeto Delphi XE 2.

Desenvolvimento OO no Delphi XE 2:

A orientação a objetos é ensinada nas faculdades, utilizando na grande maioria das vezes outra linguagem que não é o Delphi, mesmo assim, o que é ensinado não alcança o que um desenvolvimento real exige. Neste artigo vamos abordar a orientação a objetos como um todo e veremos através de um exemplo como o Delphi XE 2 nos ajuda agora a aplicá-la.

O desenvolvimento orientado a objetos fascina muito desenvolvedor Delphi, mas às vezes por insegurança ou falta de prática mesmo, não o aplica em sua totalidade. Muitos só viram orientação a objetos na faculdade, e em outra linguagem. Ainda existe a situação onde muitos desenvolvedores iniciantes nem se dão conta que um formulário é uma classe que deriva de TForm (classe base de formulários), que um componente TButton é uma classe também e assim por diante.

O Delphi já nasceu orientado a objetos, mas sua natureza RAD acaba escondendo sua estrutura orientada a objetos. É muito mais prático arrastar e soltar do que criar uma estrutura que separe a UI das regras de negócio. A partir do Delphi XE 2 isso começa mudar. Recursos como o LiveBinding tiram do meio do caminho o grande empecilho e desculpa que havia: como vou ligar meus objetos à tela? Agora é possível utilizar classes de negócio (que entenderemos no decorrer do artigo) como fonte de dados reais.

Para se desenvolver um bom software orientado a objetos devem-se seguir algumas boas práticas, estratégias e organização que veremos a seguir e só então partiremos para a construção do exemplo.

Princípios de um sistema orientado a objetos

Sempre que encontramos um sistema de muita manutenção tentamos resolver seus problemas imaginando que uma melhor coleta de dados junto ao usuário resolva o problema e o sistema se torne mais fácil depois disso. Existem problemas que vão além disso, problemas esses que estão localizados na estrutura do projeto. Existe um fato único que não muda: requisitos de sistema sempre mudam. Assim, é melhor preparar o desenvolvimento para isso, melhorar a coleta de informações ajuda, mas não resolve. O desenvolvedor deve criar o pensamento de "como escrever um código que se adapte fácil e rapidamente às mudanças que os clientes solicitam". Para ajudar, existem alguns princípios:

no Delphi XE 2

PAULO QUICOLI

Modelar para interfaces, favorecer composição sobre herança e encontrar o que varia, e encapsulá-lo.

Modelar para interfaces

Existem dois problemas que impedem um sistema de ser facilmente alterado quando há uma mudança em seus requisitos: baixa coesão e alto acoplamento.

• Baixa coesão: Isso é detectado quando encontramos rotinas/classes que realizam várias funções e essas são dependentes entre si, ou seja, é o quanto as operações de uma rotina/classe dependem de outras;

• Alto acoplamento: Encontramos um alto acoplamento quando temos rotinas/classes que são muito dependentes de outras, aonde um alteração nessa conduz a um cascateamento de alterações, ou seja, é o quanto as rotinas/classes estão dependentes uma das outras.

Para amenizar, ou evitar esses problemas, pode-se aplicar o conceito de modelar para interfaces. Interface aqui não é a interface gráfica com o usuário do seu sistema, e sim o conjunto de operações, propriedades e atributos que compõe uma determinada classe.

É através de sua interface que uma classe se faz conhecida e interage com outras classes. Para aplicar esse conceito, podem-se utilizar heranças iniciem a partir de classes abstratas, que nada mais são do que interface pura, já que seus métodos são implementados por seus descendentes, ou o tipo interface mesmo que oferece vantagens como possuir um contador de referências que, evita falhas no gerenciamento de memória.

Favorecendo composição sobre herança

Em sistemas orientados a objeto, para se conseguir reutilização de funcionalidades, são utilizadas duas técnicas em especial: herança e composição. A herança permite que o interior de uma classe base seja vista pelas suas subclasses, isso caracteriza a herança como uma técnica white box, ou "caixa branca".

A composição concede reutilizações mais complexas e exige que os objetos envolvidos possuam interfaces bem elaboradas. Quando a composição é utilizada em alternativa à herança, o interior dos objetos não fica visível entre si e devido a isso é caracterizada como black box, ou "caixa preta".

A herança permite que de forma fácil seja alterada a implementação de uma funcionalidade. Isso é obtido através da técnica de sobrecarga de operações. Um problema com esse tipo de reutilização está no fato de ser resolvido em tempo de compilação, levando certo enrijecimento ao modelo, porque em tempo de execução não é permitido alterar qualquer implementação de operações de uma classe base.

Os mais puristas em orientação a objetos dizem que a principal desvantagem da herança está na exposição do interior de uma classe base à suas subclasses, quebrando assim o conceito de encapsulamento. Essa exposição conduz a um modelo de dependência de implementações entre as classes envolvidas, isso porque a parte exportada da classe base faz parte também das subclasses.

Qualquer alteração na classe base implica em alterações nas subclasses, o que não é recomendado quando se deseja um modelo flexível. A composição de objetos exige que as interfaces sejam respeitadas, consequentemente devem ser bem planejadas para que não haja impedimentos de reutilização, isso porque a implementação de composição é feita através de referências.

Algumas desvantagens vistas na herança, como quebra de encapsulamento e dependência de implementação, não são vistas na composição, isso porque os objetos envolvidos são acessados somente por suas interfaces.

O impacto disso na modelagem, é que classes manterão seu encapsulamento e estarão voltadas para sua tarefa apenas, porque não há o que herdar de outras classes, obedecendo assim o conceito de alta coesão.

Porém, como nada é perfeito, o uso de composição prende o comportamento do modelo aos relacionamentos entre os objetos, que pode implicar em maior complexidade.

Encontrar o que varia e encapsular

Tradicionalmente o encapsulamento é visto apenas como "ocultação de dados", porém é mais do que isso, e para exemplificar vamos utilizar a seguinte analogia: "Tomando o conceito de que um guarda-chuva é algo que nos protege da chuva, um automóvel também poderia ser chamado do guarda-chuva". Essa visão não está errada, porém limita o que é um automóvel, da mesma forma acontece com encapsulamento. Podemos encapsular além de dados, qualquer implementações, classes derivadas e comportamentos.

Com o conceito de encapsulamento ampliado pode-se aplicar uma melhor forma de decomposição. No livro Design Patterns: Elements of Reusable Object-Oriented Software, é sugerido o seguinte:

"Considere o que deveria ser variável em seus sistemas. Essa abordagem é o oposto ao focar na causa da remodelagem. Ao invés de considerar o que pode forçar uma mudança na

modelagem, considere o que você deseja que seja hábil de mudar sem uma remodelagem. O foco aqui está em encapsular o conceito que varia".

A aplicação desse conceito leva ao desenvolvimento de camadas entre os objetos, diminuindo o acoplamento. Para visualizar melhor esse conceito, que pode ser complexo inicialmente, quero demonstrar com algo do dia a dia. Vamos imaginar um sistema de Log. Ao analisar o negócio você identifica a possibilidade de várias formas do registro do log. Para começar a classe TLog é criada. A grande questão é, como gerenciar essa variação de formas de registro desse TLog. Existem três formas, duas podem ser chamadas de "tradicionais", enquanto que a última faz uso do conceito de encapsulamento de comportamento.

Primeira forma

Uma solução rápida (porque não dizer RAD) seria simplesmente inserir em sua classe TLog um flag FormaDeArmazenamento que representaria os valores possíveis. Esta implementação vai conduzir todas as operações que envolvam a rotina de log a utilizar "ifs" para detectar qual a forma atual. E quando uma nova forma de armazenamento surgir? Todos esses "ifs" deverão ser revisados, gerando um trabalho desnecessário. Esta é uma solução muito utilizada por desenvolvedores menos experientes, que comumente não conseguem prever o impacto de uma alteração, seja por desconhecimento de regra de negócio ou por inexperiência mesmo.

Segunda forma

Esta forma é muito utilizada por desenvolvedores que já conhecem algo de orientação a objetos. Nela montamos uma hierarquia de classes representando cada tipo de log. Assim, teríamos uma classe base TLog com os atributos comuns a qualquer log e para as variações teríamos classes específicas que herdam de TLog, teríamos TLogArquivoTexto, TLogBancoDados e TLogArquivoXML e assim por diante. Todo esse detalhamento pode ser considerado excessivo, visto que Classes que pretendem usar TLog deverão ter conhecimento também de suas subclasses, já que é nelas que se encontra a informação da forma de armazenamento.

Terceira forma

Como já dito, esta faz uso do conceito de encapsular o comportamento do que varia. Assim, foi identificado o que varia: as formas de armazenamento, e isso é então encapsulado em uma classe abstrata TFormaDeArmazenamento. As variações são então implementadas em classes específicas (TArquivoTexto, TBancoDeDados e TArquivoXML. etc). A classe TLog por sua vez possui uma referência à classe abstrata TFormaDeArmazenamento que expõe uma interface comum, ou seja, ela não conhece nada sobre como será salvo o log, pois TFormaDeArmazenamento é abstrata, e delega aos seus descentes a implementação. Se um novo formato de armazenamento surgir, basta implementar herdando da classe abstrata, não será necessário modificar a classe TLog para nada, como pode-se ver, ela está desacoplada das implementações, do que varia, gerando assim menos manutenção. Assim, concluímos que:

- Classe abstrata: É uma abstração que conceitua todas as variações de uma classe;
- O que é comum: Define classes abstratas;
- Variações: Definem as variações das classes abstratas, sua implementação;
- Especificação: Definição de interfaces que permita o uso e atenda as variações.

Padrões de Projeto

Não é possível citar desenvolvimento orientado a objetos sem mencionar os padrões de projeto. Os padrões de projeto (Design Patterns) surgiram de estudos na área de arquitetura civil e foram posteriormente aplicados na área de análise e desenvolvimento de software, sendo considerados como boa prática. Segundo Martin Fowler (grande nome na área de orientação a objetos), "um padrão é uma ideia que tem sido útil em um contexto prático e que provavelmente será útil em outros", ou seja, é uma solução já testada para problemas de modelagem que acontecem sempre. Talvez quem não utilize a orientação a objetos na prática está se perguntando agora que problemas são esses. Alguns deles foram apresentados nos parágrafos anteriores, como baixa coesão, alto acoplamento e pouca flexibilidade. Os padrões são definidos como boas práticas justamente porque seguem os princípios explicados até agora. O uso desses padrões traz benefícios para um time de desenvolvimento. O primeiro é que ele cria uma linguagem comum entre os desenvolvedores no que tange aos problemas encontrados e soluções. Para que um desenvolver explique a outro como resolver um problema X ele vai responder apenas "use o padrão Y", e automaticamente o desenvolvedor com problemas irá entender do que se trata. É uma linguagem comum entre eles, como um jargão. Outro benefício está na própria estrutura de um padrão de projeto. Uma solução só pode ser considerada como padrão se: identificar o erro onde se aplica, explicar os efeitos da solução proposta, explicar conceitualmente a solução proposta, exibir alguma codificação da solução e acima de tudo, ter sido aplicada e testada por várias situações. Esses padrões foram classificados e catalogados quanto ao tipo de problema que resolvem. Temos as seguintes categorias: Criacionais, Estruturais e Comportamentais.

Criacionais

Os padrões dessa categoria têm a característica de abstrair a instanciação e o processo de instanciação de objetos, de tal forma que um sistema pode não depender (saber) como são criados determinados objetos. Para conseguir essa independência e flexibilidade, os padrões criacionais encapsulam as classes concretas que existem. Nessa categoria existem cinco padrões mais comuns:

- Abstract Factory;
- Builder;
- Singleton;
- Prototype;
- Factory Method.

Estruturais

Os padrões estruturais flexibilizam a criação de classes e objetos, permitindo que classes independentes trabalhem juntas. Isso é conseguido através do uso de herança para compor interfaces ou implementações. Nessa categoria temos:

- Adapter;
- Composite;
- Proxy;
- Facade;
- Bridge;
- Decorator.

Comportamentais

Gerenciam algoritmos e responsabilidades de objetos. Temos os seguintes padrões comportamentais:

- Chain of responsability;
- Command;
- Interpreter;
- Iterator;
- Mediator;
- Memento;
- Observer;
- State;
- Strategy;
- Template method;
- Visitor.

Contudo, o desenvolvimento orientado a objetos não deve ser uma busca pelos padrões de projeto, eles devem ser utilizamos apenas quando os problemas realmente acontecem.

Nota

Como ficaria muito extensa a demonstração e explicação neste artigo sobre cada padrão, na seção Links encontra-se um endereço que explica e demonstra cada um dos mesmos.

Persistência dos objetos

Há um salto enorme de conceito entre Programação Procedural e Programação Orientada a Objetos. O primeiro grande passo a se tomar é entender que um sistema OO nasce de classes e não do banco de dados como estamos acostumados a fazer em nosso dia-a-dia. O segundo é, como salvar os objetos dessas classes em um banco relacional. É disso que trata a persistência de objetos. Solucionar essa dúvida implica entrar em dois assuntos, mapeamento objeto relacional e estratégia de persistência. Abordemos então o primeiro assunto.

Mapeamento e desencontro objeto relacional

De forma resumida, a tarefa de se dizer a um sistema onde determinada classe será salva em um banco de dados relacional, como uma herança pode ser representada ou como identificar um objeto dentro do SGBD é chamada de mapeamento objeto relacional. Existe uma diferença entre classes e tabelas, a essa diferença dá-se o nome de desencontro objeto-relacional ou impedância objeto-relacional.

Quando se programa pensando em objetos levamos em conta seu comportamento e seus dados, diferentemente de um desenvolvimento centrado a dados, onde levamos em conta apenas a melhor forma de se armazenar os dados e a lógica do negócio envolvido fica separada dos dados.

Mesmo tendo um banco de dados relacional podemos unir dados e comportamento em classes, isto através do mapeamento relacional. Para conseguir isso temos algumas estratégias que mesclam os dois mundos, a principal estratégia é o conceito de OID, que nada mais é do que um identificador único para os objetos.

A importância dos OIDs

Para minimizar o impacto do desencontro objeto-relacional, temos que manter um paralelo em alguns aspectos aos bancos de dados relacionais. Como nossos objetos serão salvos em um banco relacional devemos disponibilizar uma forma de encontrá-los. Um OID identifica um único objeto em meio a tantos outros, ele representa a "chave-primária" do nosso banco relacional.

O uso de OIDs facilita a busca de objetos e também o relacionamento entre os mesmos. Isso porque quando todas as tabelas utilizam o mesmo tipo de coluna para manter suas chaves, o desenvolvimento de código genérico para lidar com elas é facilitado.

Existem algumas considerações sobre as características de um OID:

- Um OID não deve ter significado para as regras de negócio: Existe um fato que pode passar despercebido ao modelarmos nossas tabelas. Qualquer coluna de uma tabela que tem significado para as regras do negócio, pode sofrer mudanças, isso porque regras de negócio mudam. Colocar campos que contém significado como chaves é um erro que deve ser evitado. Imaginem a situação onde para identificar uma venda é utilizado como chave primária um número que representa a Nota Fiscal. Inicialmente o número da nota fiscal contém apenas números, porém mudam-se as regras de negócio e agora o número de uma NF pode conter letras também. O tipo da coluna que armazena esse valor deverá ser mudado em todas as tabelas, seja onde é chave primária seja onde é chave estrangeira. Uma simples mudança desse tipo torna-se um pesadelo. Sendo assim um OID não deve ter significado algum para os negócios, ele deve ser apenas um número sequencial que identifica um único objeto em meio a outros;
- Nível de unicidade de um OID: Outro fator que deve ser considerado ao utilizar OIDs, sua unicidade. Ele pode ser único para o universo todo de objetos, ou pode ser único apenas em meio à classe que pertence. Vamos explicar melhor. Você pode ter, por exemplo, um objeto Aluno que recebe um OID número 5400. Esse número é único em meio a todos os registros do banco de dados. Não existe outro registro com esse OID. Agora, o número 5400 pode também ser único apenas entre os objetos de tipo Aluno, existirão outros registros com esse número, porém pertencerão a outras tabelas/classes. Ao utilizar OIDs deve-se escolher entre essas duas opções, levando em consideração que se escolher a

primeira, a coluna que conterá o valor do OID deverá suportar inteiros longos, isso porque você poderá ter bilhões, trilhões de registros em todo o banco de dados.

Mapeamento básico

Ao mapear classes para tabelas, nem sempre uma classe representa uma tabela e vice-versa. É papel do mapeamento indicar o que representa o que, só assim sua camada de persistência saberá para onde salvar seus objetos e de quais tabelas obter os dados que os compõem. Contudo, o seguinte padrão é utilizado para a maioria dos casos:

• Atributos são mapeados para colunas: Um atributo de uma classe pode ser mapeado para 0 ou mais colunas de uma tabela. Você deve estar se perguntando por que 0 (zero)? Zero porque em uma classe nem todos os atributos são persistidos (salvos em meio físico). Um exemplo disso pode ser uma classe Aluno que possui um atributo data de nascimento e idade. Data de nascimento é persistido, porém Idade não, porque é um atributo calculado, seu valor é obtido de forma dinâmica portanto não precisa ser salvo;

• Mapeando classes para tabelas: Em situações simples geralmente classes são mapeadas diretamente para tabelas, exemplo: Classe TDespesa -> Tabela DESPESA

Porém existem situações mais complexas, por exemplo, quando nossos objetos utilizam herança para alcançar a realidade do negócio. Vamos imaginar que temos a seguinte estrutura: classe base TPessoa e classes TPessoaFisica e TPessoaJuridica que herdam de TPessoa.

Existem três formas básicas para se implementar a herança, que é um conceito de orientação a objetos, a um banco de dados relacional:

1. Uma tabela para toda hierarquia de classe: Nessa situação temos uma tabela que armazena os dados de TPessoa, TPessoaFisica e de TPessoaJuridica. Nesta situação o banco de dados fica menos normalizado, porém aqui um desempenho melhor pode ser obtido já que é menos custoso localizar as informações de um determinado veículo visto que a busca será feita em uma única tabela.

2. Uma tabela por classe concreta da hierarquia: Nessa implementação cada classe concreta terá sua tabela, ou seja, teremos uma tabela para armazenar a classe pessoa física e outra para pessoa jurídica. Isso pode implicar em problemas quando da inserção de novos atributos comuns às classes da hierarquia. Por exemplo, se fosse necessário adicionar uma propriedade E-Mail à classe TPessoa, no banco de dados seria necessário criar o campo Email nas duas tabelas.

3. Uma tabela por classe. Essa pode ser considerada a forma ideal, pois respeita os conceitos de normalização do mundo relacional. Nesta forma teríamos uma tabela para TPessoa, que conteria somente os campos que são comuns, outra tabela para TPessoaFisica, mas esta só com os campos pertinentes a uma pessoa física e outra para TPessoaJuridica, também com campos pertinentes a uma pessoa jurídica. Ambas tabelas podem ser ditas como especializadas e mantém uma ligação com a tabela comum através de chaves estrangeiras. Assim conseguimos chegar a uma representação relacional bem fiel de uma herança de classes. Contudo, quando existe uma hierarquia de classe muito grande, haverá mais tabelas envolvidas, que levará a vários joins entre elas, o que pode levar a uma perda de desempenho no banco de dados.

O relacionamento entre as classes e tabelas também é algo diferente. Um objeto referencia a outro simplesmente por seu endereçamento de memória, já no mundo relacional as tabelas estão ligadas por suas chaves estrangeiras/primárias. O relacionamento 1. N é o mais simples de ser expressado. Digamos que uma classe TPessoa tenha associação com TEndereco, ou seja, uma pessoa pode ter N endereços. No banco de dados isso se representa da seguinte forma: teremos duas tabelas (PESSOA e ENDERECO), e no lado N temos uma chave estrangeira que aponta para o outro lado. Teremos na tabela ENDERECO um campo apontando para a chave primária da tabela PESSOA.

Os relacionamentos M:N são representados em classes através de coleções, por exemplo, uma classe TProduto possui uma coleção de TCategoria, e essa teria uma coleção de TProduto. Já em um banco relacional teríamos uma tabela intermediária que controlaria por chaves esse relacionamento. Mas veja que para a orientação a objetos não temos o papel de uma classe intermediária.

Agenda DevMedia

Nosso aplicativo de exemplo é uma agenda simples que armazena o nome de um contato e seus telefones, e que por sua simplicidade aborda alguns dos conceitos já citados aqui. O projeto é um aplicativo FireMonkey, e nele foi adicionada uma unit ModelU.pas que é utilizada para conter nossas entidades. Entidade é um termo que é utilizado para identificar classes/objetos que possuem identidade no sistema e precisam ser persistidos. Outro termo muito comum quando se desenvolve com orientação a objetos é "Objetos de Valor", que na prática funciona como "objetos auxiliares" do modelo, que não possuem identidade e raramente precisam ser persistidos. Geralmente são valores imutáveis na regra de negócio, como um tipo conjunto (set) de meses que é utilizado apenas para exibir o mês por extenso. Foram criadas então as classes TContato e TTelefone, como mostra a **Listagem 1**.

Como ambas as classes serão salvas em um banco de dados relacional, estamos usando os conceitos de mapeamento aqui. Observe que as classes possuem um campo Id, este conterá a identificação do objeto no banco de dados. As classes também estão relacionadas entre si, em um relacionamento 1..N. Podemos dizer que um contato possui N telefones. Isso está claro na classe TContato que possui um propriedade Telefones do tipo TObjectList<T>. Esse tipo é genérico e representa uma coleção de objetos do tipo especificado. Por sua vez, TTelefone também possui uma referência de volta à TContato, assim, um telefone é de um contato (1..1).

Em TTelefone isso é expresso pela referência direta à classe TContato na propriedade Contato. Esta propriedade é o ele de ligação entre as duas classes, contudo, também temos uma pro-

priedade privada chamada ContatoId, esta é utilizada para manter a referência relacional, a chave estrangeira do banco de dados. Como é possível de perceber temos um mapeamento simples, mas que mostra como é possível transformar objetos em registros e vice versa. Essas duas classes formam o que chamamos de domínio, é o conjunto de classes que representa nosso negócio.

Listagem 1. Entidades

```
TContato = class
private
  FNome: string;
  FTelefones: TObjectList<TTelefone>;
  FId: integer;
  procedure SetNome(const Value: string);
  procedure SetTelefones(const Value: TObjectList<TTelefone>);
  procedure SetId(const Value: integer);
public
  constructor Create();
published
  property Id: integer read FId write SetId;
  property Nome: string read FNome write SetNome;
  property Telefones: TObjectList<TTelefone> read FTelefones write SetTelefones;
  procedure AdicionaTelefone(Telefone: TTelefone);
  procedure RemoveTelefone(Telefone: TTelefone);
end;

TTelefone = class
private
  FNumero: string;
  FContato: TContato;
  FId: integer;
  FContatoId: Integer;
  procedure SetNumero(const Value: string);
  procedure SetContato(const Value: TContato);
  procedure SetId(const Value: integer);
  procedure SetContatoId(const Value: Integer);
  property ContatoId: Integer read FContatoId write SetContatoId;
published
  constructor Create(Numero: string);
  property Id: integer read FId write SetId;
  property Numero: string read FNumero write SetNumero;
  property Contato: TContato read FContato write SetContato;
end;
```

Persistindo o domínio

Ao longo da Clube Delphi alguns frameworks de mapeamento objeto relacional foram apresentados, como tiOPF e Instant Objects. Mas este artigo apresenta algo novo, que ainda está desenvolvimento, mas que está estável e em uso em aplicativos reais. O DORM (Delphi Object Relational Mapper). Este é open source, possui colaboradores brasileiros e se espelha muito no NHibernate (que é uma referência no assunto).

Seu download consiste apenas do código fonte com alguns exemplos, não possui um instalador, pois não é necessário. Uma vez baixado, adicionamos no search path do projeto os seguintes caminhos vistos na **Figura 1**. Isso é necessário para que o projeto encontre as classes do DORM.

O mapeamento em si é feito através de um arquivo texto no formato JSON. O JSON (JavaScript Object Notation) é uma espécie de padronização para transporte de informações entre ambientes heterogêneos. Bem, padrão, transporte, ambiente heterogêneo, tudo isso nos remete ao XML, então vem a questão: Por que JSON e não XML? O principal motivo seria pelo fato da notação JSON ser extremamente mais simples do que a XML. Não existem tags diversas nem regras complexas para escrita e muito menos mecanismos complexos de parser. Para se ter uma ideia observe a **Figura 2**, ela mostra a sintaxe para notação JSON para um objeto.

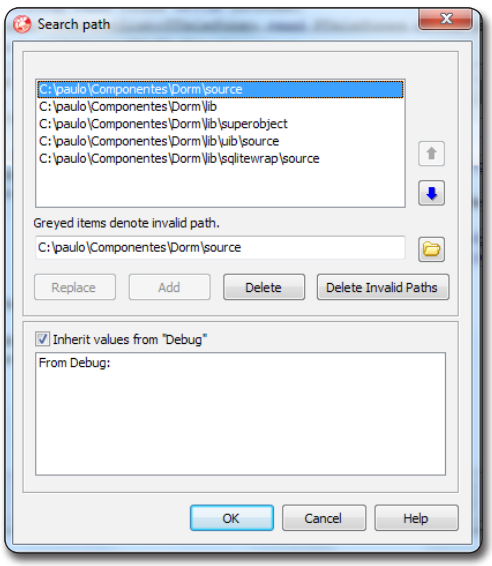


Figura 1. Search Path

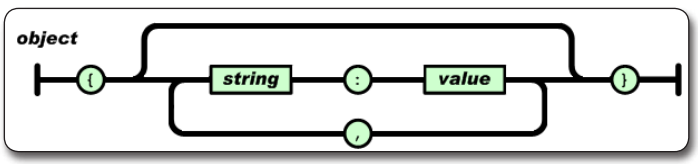


Figura 2. Representação JSON

Sendo assim, simplesmente abrimos o bloco de notas, criamos um arquivo de nome qualquer (no exemplo foi utilizado o nome dorm_firebird_uib.config) e nele incluímos o mapeamento da **Listagem 2**, que é explicado na sequência.

Esse arquivo é representação em JSON de um objeto Persistence do DORM, que é responsável pela configuração geral do mecanismo de persistência. Observe e veja que temos a possibilidade de configurar três ambientes distintos para efetuar uma conexão com um banco de dados. Temos development, test e release. Essas especificações são utilizadas para determinar bancos de dados específicos com o cenário. Assim, se estamos em desenvolvimento, podemos usar um banco de dados X, em release um Y.

A propriedade database_adapter indica qual banco de dados usaremos, no caso foi escolhido o Firebird. O DORM fornece duas formas de acesso ao Firebird, uma usando DBX e outra UIB, como pode ser reconhecido, estamos utilizando a opção com os componentes UIB (que estão inclusos no pacote do DORM).

Continuando, temos a string de conexão, um detalhe aqui é a necessidade de se informar as barras de caminho de forma dupla

Listagem 2. Mapeamento DORM

```
{
 "persistence": {
 "development": {
  "database_adapter": "dorm.adapter.UIB.Firebird.TUIBFirebirdPersistStrategy",
   "database_connection_string":"C:\\Users\\paulo\\Documents\\RAD Studio
    \\Projects\\Contatos\\Banco\\contatos.fdb",
   "username": "sysdba",
  "password":"masterkey",
   "keys_generator":"dorm.adapter.UIB.Firebird.TUIBFirebirdTableSequence",
   "key_type":"integer", // "string" | "integer"
   "null_key_value": 0},
"test": {
   "database_adapter": "dorm.adapter.UIB.Firebird.TUIBFirebirdPersistStrategy",
   "database_connection_string":"localhost:C:\\DEV\\dorm\\samples\\data\\
    dorm_test.fdb",
   "username": "sysdba",
   "password":"masterkey",
   "keys_generator":"dorm.adapter.UIB.Firebird.TUIBFirebirdTableSequence",
   "key_type":"integer", // "string" | "integer"
   "null_key_value": 0},
  "release": {
   "database_adapter": "dorm.adapter.UIB.Firebird.TUIBFirebirdPersistStrategy",
   "database_connection_string":"localhost:C:\\DEV\\dorm\\samples\\data
  \\dorm_prod.fdb",
   "username": "sysdba",
   "password":"masterkey",
   "keys_generator":"dorm.adapter.UIB.Firebird.TUIBFirebirdTableSequence",
   "key_type":"integer", // "string" | "integer"
   "null_key_value": 0},
   "persistent_classes": [
         "TContato", "TTelefone"
         ]
  },
 "config": {
   "logger_class_name": "dorm.loggers.CodeSite.TdormFileLog"
 },
 "mapping":
   {

         "TContato":
    {
      "package":"ModelU",
      "table": "CONTATO",
      "id": {"name":"Id", "field":"ID_CONTATO", "field_type":"integer"},
      "fields":[
                  {"name":"Nome", "field":"NOME", "field_type":"string", "size": 100},
                ],
      "has_many":[{
        "name": "Telefones",
        "class_name":"TTelefone",
        "child_field_name":"ContatoId",
        "lazy_load": false
      }]
    },
         "TTelefone":
    {
      "package":"ModelU",
      "table": "TELEFONE",
      "id": {"name":"Id", "field":"ID_TELEFONE", "field_type":"integer"},
      "fields":[
                  {"name":"Numero", "field":"NUMERO", "field_type":"string", "size": 20},
                  {"name":"ContatoId", "field":"ID_CONTATO", "field_type":"integer"}
                ]
       }
   }
}
```

"\\" devido a implementações do framework. Temos username, password e Keys_generator. Essa propriedade indica como será feita a geração das chaves primárias. Foi informado o formato TableSequence que faz uso de generators nativos do banco de dados para cada classe. Assim, se lembrarmos novamente das regras de mapeamento, o DORM está empregando a ideia de unicidade dentro de uma área, e não do todo. Poderemos ter objetos de classes diferentes com o mesmo Id, bem similar ao que já fazemos no mundo relacional.

Temos ainda a propriedade Key_Type, que representa de que tipo de dado são feitas as chaves (string ou integer) e ainda que valor indica que uma chave é vazia (propriedade null_key_value).

Já a propriedade persistent_classes é um array com os tipos de classes que deverão ser persistidas. As classes que não constarem nessa lista, não são salvas. Depois temos a definição de um objeto config. Nele é possível utilizar o CodeSite que é disponível no Delphi XE 2 para que sejam exibidas informações de tudo que é feito pelo DORM em tempo de execução. Essa é uma ótima forma de visualizar os comandos SQL gerados e detectar qualquer problema neles.

Agora é hora de examinarmos como uma classe é mapeada. Como se vê no arquivo, temos a definição de um objeto mapping. Nele temos a declaração do tipo TContato, onde especificamos na propriedade package em qual unit o mesmo está localizado, qual será o nome da tabela (table) e que propriedade da classe representa a chave primária na tabela informada (propriedade id). Observe que podemos ter propriedades na classes com um nome, e, tabelas e colunas de nomes diferentes. Essa é uma das funções do mapeamento. Após especificar a chave, temos uma propriedade fields que um array do restante das propriedades da classe a ser persistida. Se você não deseja que uma determinada propriedade não seja persistida, basta não relacioná-la. Informamos o nome da propriedade na classe (name), qual campo da tabela que representará essa propriedade (field), seu tipo na classe (field_type), e o tamanho que o campo terá (size). Para campos inteiros não informamos tamanho algum. Na sequência do mapeamento da classe TContato informamos que ela possui um relacionamento do tipo 1..N, através da declaração do objeto has_many. Esse também é um array onde configuramos o relacionamento das classes. Dizemos em name qual propriedade representa a ligação, com qual classe a ligação é feita, qual é a propriedade na classe filha que contém a chave dessa ligação (por isso que definimos em TTelefone uma propriedade privada ContatoId) e se para carregar essa classe é utilizado o esquema de lazy load (lazy_load). Lazy load, ou carga tardia, é um conceito onde uma coleção de objetos só é trazida quando a mesma for utilizada.

O mapeamento da classe TTelefone é muito similar ao TContato, sendo até mesmo mais simples. Feito o mapeamento, o arquivo deve ser salvo na pasta onde o executável é gerado.

Interface com usuário

A interface sugerida é bem simples, temos um formulário principal que possui um menu com operações básicas, um

ListBox que exibe todos os contatos cadastrados, e um segundo ListBox que exibe os detalhes do contato, ou seja seus telefones (**Figura 3**)

Temos ainda um segundo formulário que é utilizado para realizar a inclusão de novos contatos, como mostra a **Figura 4**.

Uma vez as telas prontas, é preciso exibir os dados.

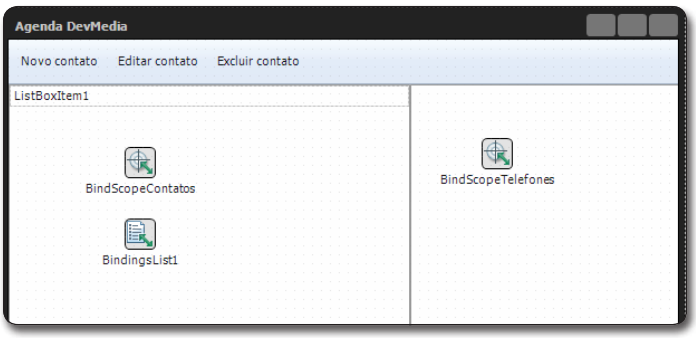


Figura 3. Formulário principal

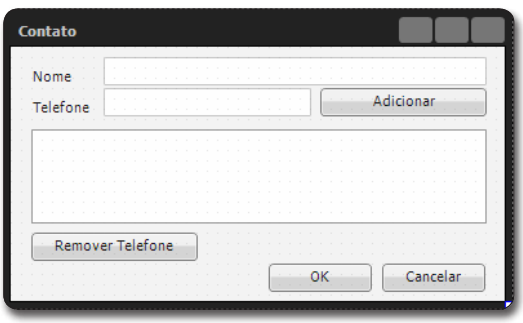


Figura 4. Manutenção de contatos

Conectando ao banco de dados

O DORM não cria o banco de dados, por isso é preciso criar manualmente o banco de dados, seguindo o que foi definido no mapeamento. O detalhe fica por conta dos generators a serem definidos, um para cada tabela. Eles devem possuir uma nomenclatura padrão: SEQ_NOMETABELA_ID, assim foram criados os seguintes: SEQ_CONTATO_ID e SEQ_TELEFONE_ID.

No formulário principal adicionamos referências às units do DORM para que possamos utilizar suas funcionalidades:

```
dorm,
dorm.adapter.all,
dorm.Collections,
dorm.Commons,
dorm.Filters ;
```

O DORM emprega o conceito de sessão, que representa uma conexão com o banco de dados, assim, declaramos no formulário principal um campo privado Session: TSession. Nele também estão centralizadas as funcionalidades para salvar, recuperar, excluir ou atualizar objetos. No construtor do formulário principal nós instanciamos a sessão, indicando o nome do arquivo e qual ambiente utilizar (desenvolvimento, teste ou release):

```
ConfigFileName := 'dorm_firebird_uib.conf';
Session := TSession.CreateConfigured(TStreamReader.Create(ConfigFileName),
  deDevelopment);
CarregarDados;
```

O método CarregarDados foi criado para realizar a consulta de todos os contatos e atualizar a exibição dos mesmos. Aqui quero particionar a explicação desse método pois ele envolve duas coisas diferentes. Primeiro, para trazer os objetos do banco de dados também foi utilizado o objeto Session:

```
Session.FillList<TContato>(ColecaoContatos);
```

O método FillList é genérico, e nele passamos qual tipo que vamos carregar do banco de dados, o resultado é inserido um objeto TObjectList<T> que deve ser do mesmo tipo passado ao FillList. No caso, ColecaoContatos é uma variável pública definida na unit ModelU.pas. Em segundo lugar temos a configuração o LiveBinding.

Até a versão XE2 do Delphi, foi um problema trabalhar com objetos de negócio, visto que não era fácil realizar sua apresentação na tela. O Delphi não dispunha de mecanismos automáticos, seus controles visuais eram prontos para operar com DataSets. Com a inclusão do LiveBinding essa limitação acabou.

Como pode ser visto na **Figura 3** foram adicionados alguns ao formulário principal: BindScopeContatos, BindingList1 e BindScopeTelefones. Os controles BindScope determinam (como seu nome diz) o escopo do binding, ou seja, qual é o objeto que deverá ser ligado a algum controle visual. Já o BindingList1 contém as definição das ligações. Ao efetuarmos um clique duplo sobre ele, a janela da **Figura 5** é exibida.

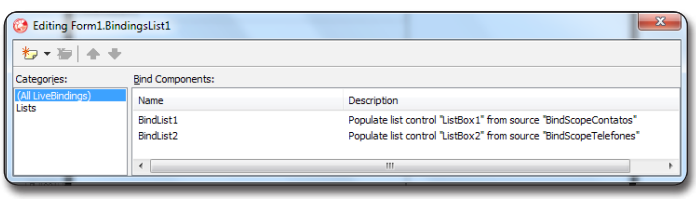


Figura 5. Lista de bindings

Cada definição de binding foi adicionado pelo botão New Binding e então escolhida a opção TBindList. Nela podemos ligar uma coleção de objetos a controles visuais através das propriedades SourceComponent (que é o BindScope) e ControlComponent (que é o componente que irá exibir a informação). Uma vez definidas essas propriedades via Object Inspector é preciso definir quais propriedades dos controles serão utilizadas para exibição dos dados. Essa configuração é feita através de um clique duplo no BindList para que se apresente sua janela de configuração **(Figura 6)**.

Definimos que a propriedade Text do ListBox1 irá exibir o valor do objeto corrente em BindScopeContatos. Observe que a sintaxe é Current.Nome, onde Current é uma instância do objeto que você

definir. Tudo isso é avaliado em tempo de execução e não em projeto. Assim, o método CarregarDados pode continuar e atualizar o databinding:

```
BindScopeContatos.Active := false;
BindScopeContatos.DataObject := ColecaoContatos;
BindScopeContatos.Active := true;
BindList1.FillList;
```

Passamos para o BindScopeContatos a coleção de objetos que temos, e o método FillList do BindList1 se encarrega de ler os objetos e exibi-los. Também é preciso exibir os telefones de um contato ao selecioná-lo na lista. No evento OnClick do ListBox1 temos o código da **Listagem 3**.

Listagem 3. Carregando os telefones

```
if ListBox1.ItemIndex > -1 then
begin
  BindScopeTelefones.Active := False;
  Session.LoadRelations(ColecaoContatos[ListBox1.ItemIndex]);
  BindScopeTelefones.DataObject := TContato(ColecaoContatos[ListBox1.ItemIndex]).Telefones;
  BindScopeTelefones.Active := True;
  BindList2.FillList;
end;
```

Listagem 4. Excluindo um objeto

```
if ListBox1.ItemIndex > -1 then
 begin
  contato := ColecaoContatos[ListBox1.ItemIndex];
  Session.Delete(ColecaoContatos.Extract(contato));
  BindScopeTelefones.Active := false;
  BindScopeTelefones.DataObject := nil;
  BindScopeTelefones.Active := true;
  BindList2.ClearList;
  BindList1.FillList;
 end;
```

Listagem 5. Criando um novo contato

```
procedure TForm1.menuNovoClick(Sender: TObject);
var
  contato: TContato;
  I: Integer;
begin
  CadContatoF := TCadContatoF.Create(nil);
  CadContatoF.ShowModal;
  if CadContatoF.Confirmou then
  begin
    contato := TContato.Create;
    contato.Nome := CadContatoF.Edit1.Text;
    for I := 0 to CadContatoF.ListBox1.Items.Count-1 do
    begin
      contato.AdicionaTelefone(TTelefone.Create(CadContatoF.ListBox1.Items[i]));
    end;
    Session.Save(contato);
    CarregarDados;
  end;
  FreeAndNil(CadContatoF);
end;
```

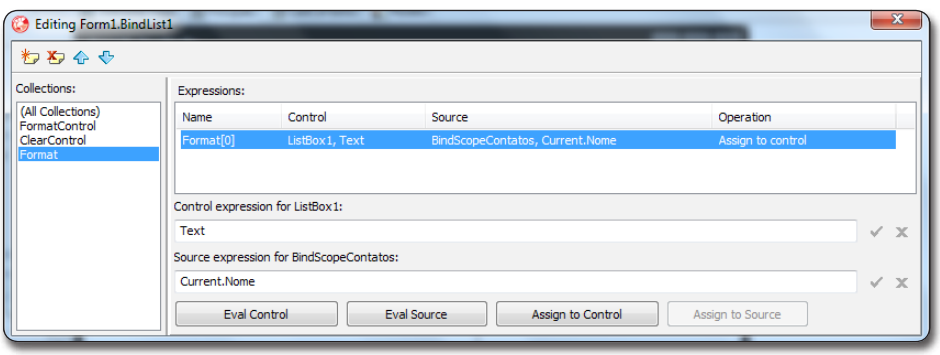


Figura 6. Configuração do Binding

O método LoadRelations de Session recebe como parâmetro um objeto, este foi obtido diretamente da coleção de objetos. Assim, Session efetua uma consulta no banco de dados e carrega os relacionamentos que o objeto passado possui com outros objetos. Isso é feito baseando-se no mapeamento feito anteriormente. Se você inserir manualmente alguns registros no banco de dados e executar a aplicação, verá os resultados.

A exclusão também é feita pelo objeto Session, como é visto a seguir na **Listagem 4.**

Pelo método Delete de Session passamos o objeto que desejamos excluir, observe que foi executado já o método Extract de TObjectList para que o objeto saia da lista. Uma vez que o objeto saiu da lista, não é necessário mais exibir seus telefones, visto que isso mantê-los em exibição é um erro, pois na realidade eles não existem mais no banco de dados. Então simplesmente limpamos o BindList2. O DORM cuida da manutenção do relacionamento das classes, ao excluir um contato, automaticamente ele exclui seus telefones.

Para incluir um novo contato também é utilizado o objeto Session. Aqui simplesmente pegamos os valores informados na tela de cadastro e instanciamos um novo objeto TContato, passando os valores obtidos (**Listagem 5**).

O método Save de Session se encarrega de adicionar o novo objeto TContato no banco de dados. O interessante é que como se vê, você não precisa salvar os objetos TTelefone contidos em TContato, o DORM automaticamente cuida disso para você.

Conclusão

Empregamos neste artigo conceitos de mapeamento objeto relacional utilizando um novo framework chamado DORM que, merece um artigo dedicado somente a ele por sua infinidade de recursos. Utilizamos também os recursos do Delphi XE 2 que permitem o uso de objetos de forma ágil no que tange à apresentação dos mesmos, contudo este exemplo pode ser melhorado e muito. Pode-se empregar uma separação de camadas lógicas e extrair do formulário principal a utilização do objeto Session, centralizando essas operações de CRUD

(manipulação de dados) em uma classe que segue o padrão Repository. Esse padrão determina que crie um repositório para as classes que serão persistidas e elas assim oferecem métodos como Salvar, Excluir, Atualizar, isolando outras classes da real implementação desses métodos.

No exemplo ainda faltou a parte de edição dos contatos, deixo esta tarefa com você leitor, para que explore o DORM e conheça mais sobre ele. Abraço.

DORM
http://code.google.com/p/delphi-orm/

DoFactory – Padrões de Projeto implementados e catalogados
http://www.dofactory.com/Default.aspx

Paulo Quicoli
pauloquicoli@gmail.com
Editor da revista ClubeDelphi e editor técnico .Net Magazine. Formado em processamento de dados pela FATEC-TQ. Atua como analista programador de sistemas e coordenador técnico da equipe de desenvolvimento .NET da Siplan Control-M (www.siplancontrolm.com.br), e professor da FATEC, campus Taquaritinga. Blog: http://quicoli.worpress.com

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

CURSOS ONLINE

A Revista .net Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

.net magazine

MAIS de 25 CURSOS DISPONÍVEIS

CONHEÇA OS CURSOS MAIS RECENTES:

- Curso .net Entity Framework - Loja Virtual
- Novidades do Visual Studio 2010
- Componentes DevExpress
- Crie um chat utilizando Asp.Net 3.5 e Linq
- Novidades do C# (csharp) 3.0
- POO com C#
- Microsoft Visual Studio Tools for Office (VSTO)

Para mais informações :
www.devmedia.com.br/curso/netmagazine
(21) 3382-5038

DevMedia

Design Patterns
Utilizando os padrões Factory Met

Resumo DevMan

De que trata o artigo:

Este artigo procura abordar a utilização de padrões de projeto (design patterns) em soluções de software, demonstrando para isto como os patterns Factory Method e Template Method podem ser empregados na elaboração de sistemas com uma arquitetura flexível e bem definida.

Em que situação o tema é útil:

Design patterns são técnicas pré-definidas e empregadas na resolução de problemas em atividades que envolvam a construção de softwares. O pattern Factory Method é uma alternativa que pode ser empregada na geração de novas instâncias de tipos que contam com uma estrutura em comum, sem contudo obrigar as classes consumidoras a implementarem as condições que levam à obtenção dos objetos esperados. Já o padrão Template Method é uma opção para situações que envolvam variações dentro de um algoritmo de processamento, nos quais determinadas etapas podem apresentar comportamentos próprios decorrentes de algum tipo de necessidade específica.

Design Patterns - Utilizando os padrões Factory Method e Template Method:

É extremamente comum, no decorrer de uma série de projetos, a repetição de determinadas atividades que requerem um tipo de tratamento particular. Em diversos casos é possível que outros profissionais já tenham se deparado com situações semelhantes e, a partir disso, disponibilizado meios que buscam resolver estas questões. Tais soluções são conhecidas como design patterns (padrões de projetos), focando normalmente na resolução de problemas específicos e recorrentes dentro do desenvolvimento de software. O pattern Factory Method é uma alternativa que pode ser empregada para a instanciação de objetos de diferentes tipos, mas que contam com uma estrutura comum. Já o padrão Template Method permite o controle de comportamentos variáveis dentro do algoritmo que corresponde às ações desencadeadas por um método, fazendo uso para isto de subclasses que estendem tipos abstratos.

É um fato dado como certo que mudanças acontecerão ao longo do ciclo de vida de uma aplicação de software. Alterações em atendimento a uma nova necessidade da organização ou mesmo a evolução de recursos pré-existentes são aspectos com os quais os profissionais elencados em projetos de sistemas sempre irão se deparar. Novas soluções de software também são criadas a todo o momento, como um reflexo das constantes mudanças a que estão sujeitas muitas companhias.

Design patterns não são conceitos que se encontram ligados a uma tecnologia específica. Como estão voltados à solução de problemas recorrentes em projetos de software, representando soluções concisas e que priorizam a obtenção de código bem estruturado, muitos padrões podem ser aplicados a aplicações concebidas nas mais diferentes linguagens de programação (Delphi, C#, VB.NET, Java, C++ etc.).

Considerando a ampla diversidade de tipos de aplicações de software existentes atualmente, não é difícil de imaginar que tenham sido desenvolvidos patterns para os mais diversos contextos. Os esforços para a definição de padrões iniciaram-se na área de software ainda nos anos 90 com a publicação de 23 padrões conhecidos como GoF.

hod e Template Method

Renato José Groffe

Nota do DevMan

GoF é a sigla em inglês para "Gang of Four", sendo um apelido dado ao trabalho conjunto de 4 autores (Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides), os quais documentaram 23 padrões no famoso livro "Design Patterns: Elements of Reusable Object-Oriented Software". Tais patterns foram especificados na época tomando-se como base exemplos elaborados em C++ e SmallTalk; no entanto, a ampla abrangência destes padrões possibilitou que os mesmos sejam utilizados ainda hoje, auxiliando desenvolvedores Delphi, .NET e Java na resolução de uma série de problemas recorrentes em suas atividades cotidianas.

Objetivos do uso de padrões

A utilização de patterns em um projeto de software tem por principais objetivos:

- Evitar que se obtenham extensas construções de código durante a construção de uma solução;
- Possibilitar que o código obtido seja de fácil compreensão por parte de desenvolvedores que venham a trabalhar com o mesmo;
- Uma maior flexibilidade para se alterar uma aplicação quando da necessidade de se efetuarem mudanças em atendimento a requisitos de negócio;
- Divisão equilibrada de responsabilidades entre as diferentes partes que compõem um projeto;
- O reuso de componentes de software, não ficando isto restrito a uma única aplicação, mas com o reaproveitamento de funcionalidades em mais de uma solução;
- Redução eficiente da duplicação de código através de técnicas de fatoração do código, reduzindo-os a unidades pequenas e coesas.

Benefícios da aplicação de design patterns

São vantagens do uso de padrões de projetos na construção de aplicações:

- Uma maior padronização em atividades relacionadas à codificação de funcionalidades;
- A possibilidade de reuso de uma mesma arquitetura em diversos projetos;
- Uma maior coesão nos diferentes componentes de uma solução, com cada classe acumulando apenas funções relacionadas ao propósito para o qual a mesma foi criada inicialmente;
- Maior facilidade na manutenção de uma aplicação, com menores impactos quando da necessidade de se modificar algo, considerando para isto um baixo acoplamento entre os diversos componentes que constituam a solução;
- A redução de custos em novos projetos, a partir do reaproveitamento de componentes e funcionalidades já elaborados anteriormente;
- A obtenção de soluções concebidas segundo boas práticas de OO, em virtude da ênfase dada pelos diferentes design patterns à utilização de técnicas já consolidadas dentro da orientação a objetos;
- Por contarem com ampla documentação, padrões de projeto podem ser aprendidos com facilidade, fato que melhora a interação e o compartilhamento de conhecimentos entre desenvolvedores dentro de uma mesma equipe.

Tipos de patterns GoF

Tomando como base o tipo de problema a que se propõem solucionar, os patterns GoF podem ser divididos em:

- Padrões de criação: envolvem técnicas para a geração de novas instâncias de objetos;
- Padrões estruturais: estipulam formas para se trabalhar com a composição de objetos;
- Padrões comportamentais: estabelecem regras para a interação entre classes ou, ainda, definem como serão as responsabilidades das mesmas.

Na **Tabela 1** encontram-se listados os padrões GoF existentes, com os mesmos estando agrupados por tipo:

Criação de instâncias de objetos com o padrão Factory Method

Factory Method (**Figura 1**) é um padrão de criação, sendo também conhecido pelo nome de Virtual Constructor. Através deste pattern é definida uma interface para a criação de um objeto.

Tipo	Patterns
Padrões de criação	Abstract Factory, Builder, Factory Method, Prototype e Singleton.
Padrões estruturais	Adapter, Bridge, Composite, Decorator, Façade, Flyweight e Proxy.
Padrões comportamentais	Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method e Visitor.

Tabela 1. Patterns GoF classificados por tipo

No entanto, caberá às classes que implementam tal interface definir qual tipo será retornado, sendo isto normalmente feito por meio de algum parâmetro ou outra forma de configuração.

Pode-se considerar, em termos gerais, que o padrão Factory Method procura fornecer meios para encapsular a instanciação de tipos concretos. Uma classe abstrata (ClasseAbstrata no exemplo da imagem) define um método a ser implementado por suas subclasses (ClasseConcreta no exemplo), a fim de que a partir de tal operação sejam criados novos objetos de um tipo básico (instâncias de classes que implementem Produto, como ProdutoConcreto por exemplo).

Uma das vantagens deste tipo de técnica consiste, basicamente, na definição de um local para criação de instâncias dentro de um método específico para este fim, evitando-se com isso a duplicação da lógica empregada neste tipo de tarefa ao longo de um projeto. Tal fato contribui ainda para facilitar atividades de manutenção futuras: apenas o método empregado na geração de objetos seria alterado num primeiro momento.

Além disso, essa prática reduz o acoplamento entre componentes da aplicação, uma vez que os clientes (classes) passam a depender de interfaces, ao invés de tipos concretos empregados na instanciação de objetos. A ênfase na programação voltada ao uso de interfaces torna o código-fonte mais flexível a mudanças, assim como mais extensível e fácil de adaptar no futuro.

O pattern Factory Method representa a aplicação de um princípio conhecido como Inversão de Dependência. Este último enfatiza que componentes de alto nível não dependam diretamente de comportamentos de elementos de um nível inferior, mas que tal relacionamento aconteça por meio do uso de classes abstratas e interfaces, em detrimento da dependência de classes concretas. Assim, chega-se a um baixo acoplamento entre as partes que constituem uma solução, característica esta altamente desejável quando se busca a obtenção de um software dotado de uma arquitetura bem estruturada.

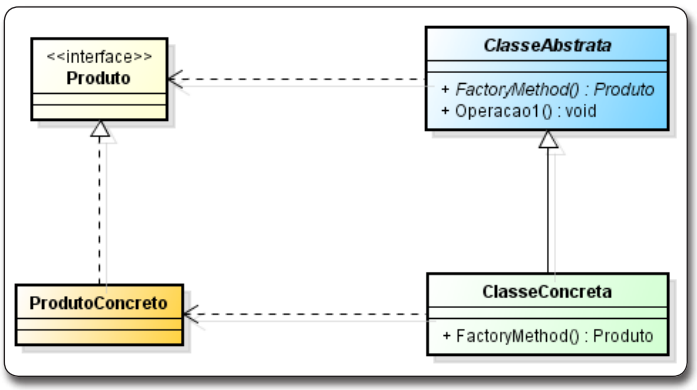


Figura 1. Representação esquemática do pattern Factory Method

Encapsulando algoritmos com o padrão Template Method

Através do padrão Template Method (**Figura 2**) define-se o esqueleto de um algoritmo que representa um fluxo de execução, sendo que alguns dos passos que compõem este último são definidos em subclasses do tipo que encapsula o algoritmo em questão. Devido a estas características este pattern é classificado como um padrão comportamental.

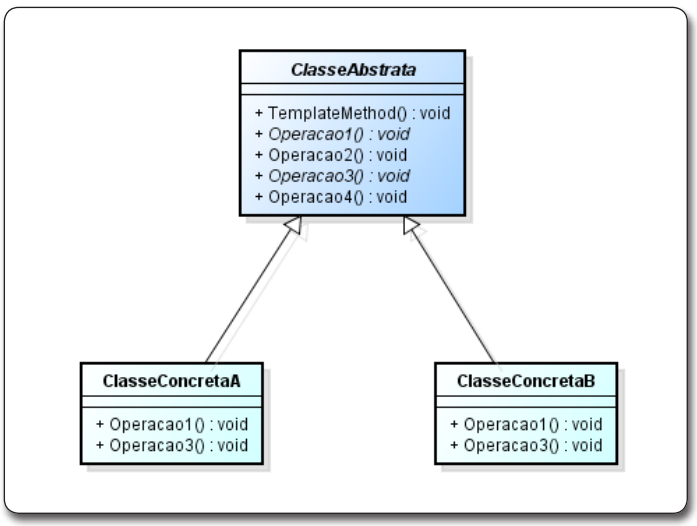


Figura 2. Representação esquemática do pattern Template Method

Um método (TemplateMethod) é criado na classe abstrata (ClasseAbstrata no exemplo da imagem) que define o uso do pattern Template Method, sendo que as diversas atividades que constituem o mesmo são implementadas em outras operações do tipo básico que se está construindo (Operacao1, Operacao2, Operacao3 e Operacao4); etapas que contem com um comportamento variável são implementadas em subclasses (ClasseConcretaA e ClasseConcretaB), sendo que as mesmas são definidas como métodos abstratos no tipo base (Operacao1 e Operacao3).

É importante ressaltar que ao se levar em conta soluções que empreguem este padrão o uso de métodos abstratos para a especificação de comportamentos que variam é a forma mais

comum, embora isto não represente a única alternativa para se atender a um determinado tipo de demanda. É possível que uma implementação padrão para uma operação encontre-se definida no tipo básico com algumas das subclasses redefinindo tal comportamento via sobrecarga do método em questão.

A adoção deste tipo de prática tem como principal benefício maximizar o potencial de reuso de um tipo, já que operações que não sofrem variações encontram-se centralizadas numa classe abstrata, ao passo que a implementação de comportamentos mutáveis é delegada a métodos abstratos que estão situadas em subclasses. Assim, esta técnica contribui para evitar a duplicação de código referente a um tipo de lógica ao longo de uma aplicação.

O padrão Template Method emprega um conceito conhecido dentro da programação de sistemas como Princípio Hollywood ("não nos chame, nós chamaremos você"): a decisão sobre o que deverá ser feito e quando cabe ao componente de nível mais alto (classe abstrata), o qual se encarrega de invocar as funcionalidades definidas nas construções de nível mais baixo (subclasses concretas).

Nota do DevMan

Os padrões Template Method e Strategy apresentam várias semelhanças, já que ambos têm por objetivo o encapsulamento de algoritmos em construções que permitam a execução de comportamentos variáveis de uma forma dinâmica. Estes patterns diferem, entretanto, na maneira como fazem isto: enquanto a utilização do padrão Strategy depende da composição, o uso de Template Method baseia-se na herança de classes.

Ainda levando em conta as características do pattern Template Method, o pattern Factory Method é visto como uma especialização do mesmo.

Exemplos de uso dos patterns Factory Method e Template Method

O desenvolvimento das aplicações demonstradas neste artigo foi realizado por meio da utilização do Embarcadero Delphi XE2, sendo que através das mesmas serão apresentadas formas de uso dos patterns Factory Method e Template Method.

Utilizando o pattern Factory Method

Para se testar o uso do pattern Factory Method deverá ser criado um projeto do tipo "VCL Forms Application" chamado TesteTemplateMethod. A partir do executável resultante serão exibidas em tela informações sobre guitarras, anos iniciais de fabricação de tais instrumentos, bem como alguns dos modelos disponíveis para cada linha.

Criação das classes que empregam o padrão Factory Method

Na **Listagem 1** está definida a interface IGuitarra, a qual determina a estrutura a ser implementada pelos diversos instrumentos instanciados por meio das classes Factory.

As classes TGibsonLesPaul (**Listagem 2**), TGibsonSG (**Listagem 3**), TFenderStratocaster (**Listagem 4**) e TFenderTelecaster (**Listagem 5**) implementam a interface IGuitarra, retornando a partir de seus diferentes métodos o nome do instrumento, o ano inicial de fabricação e alguns dos modelos mais conhecidos comercializados.

A **Listagem 6** exibe o código-fonte para a classe abstrata TGuitarra Factory. A operação CriarGuitarra é abstrata (foi marcada com as palavras-chave "virtual" e "abstract"), correspondendo ao factory method a partir do qual serão obtidas novas instâncias de tipos que implementam IGuitarra. Já o método ExibirDetalhesGuitarra aciona uma mensagem em tela em que constarão detalhes de um determinado instrumento.

Listagem 1. Interface IGuitarra

```
unit Guitarra;

interface

type
 IGuitarra = interface
  function ObterNomeGuitarra: string;
  function ObterAnoInicialFabricao: Integer;
  function ObterModelos: string;
 end;

implementation

end.
```

Listagem 2. Classe TGibsonLesPaul

```
unit LesPaul;
interface
uses Guitarra;
type
 TGibsonLesPaul = class(TInterfacedObject, IGuitarra)

  public
   function ObterNomeGuitarra: string;
   function ObterAnoInicialFabricao: Integer;
   function ObterModelos: string;
 end;

implementation
{ TGibsonLesPaul }
function TGibsonLesPaul.ObterNomeGuitarra: string;
begin
 Result:= 'Gibson Les Paul';
end;

function TGibsonLesPaul.ObterAnoInicialFabricao: Integer;
begin
 Result:= 1958;
end;
function TGibsonLesPaul.ObterModelos: string;
begin
 Result:= 'Sunburst, Goldtop, Black';

end;
end.
```

Listagem 3. Classe TGibsonSG

```
unit SG;
interface
uses Guitarra;
type
 TGibsonSG = class(TInterfacedObject, IGuitarra)
 public
  function ObterNomeGuitarra: string;
  function ObterAnoInicialFabricao: Integer;
  function ObterModelos: string;
 end;
implementation

{ TGibsonSG }
function TGibsonSG.ObterNomeGuitarra: string;
begin
 Result:= 'Gibson SG';

end;
function TGibsonSG.ObterAnoInicialFabricao: Integer;
begin
 Result:= 1961;
end;

function TGibsonSG.ObterModelos: string;
begin
 Result:= 'Black, Red';
end;
end.
```

Listagem 4. Classe TFenderStratocaster

```
unit Stratocaster;
interface
uses Guitarra;
type

 TFenderStratocaster = class(TInterfacedObject, IGuitarra)
 public
  function ObterNomeGuitarra: string;
  function ObterAnoInicialFabricao: Integer;
  function ObterModelos: string;
 end;

implementation
{ TFenderStratocaster }
 function TFenderStratocaster.ObterNomeGuitarra: string;
 begin
  Result:= 'Fender Stratocaster';
 end;

 function TFenderStratocaster.ObterAnoInicialFabricao: Integer;
 begin
  Result:= 1954;
 end;

 function TFenderStratocaster.ObterModelos: string;
 begin
  Result:= 'Sunburst, Black, Red';
 end;
end.
```

Listagem 5. Classe TFenderTelecaster

```
unit Telecaster;
interface
uses Guitarra;
type

 TFenderTelecaster = class(TInterfacedObject, IGuitarra)
 public
  function ObterNomeGuitarra: string;
  function ObterAnoInicialFabricao: Integer;
  function ObterModelos: string;
 end;

implementation
{ TFenderTelecaster }
 function TFenderTelecaster.ObterNomeGuitarra: string;
 begin
  Result:= 'Fender Telecaster';
 end;

 function TFenderTelecaster.ObterAnoInicialFabricao: Integer;
 begin
  Result:= 1948;
 end;

 function TFenderTelecaster.ObterModelos: string;
 begin
  Result:= 'Blonde, Black';
 end;
end.
```

Listagem 6. Classe TGuitarraFactory

```
unit Factory;
interface
uses Vcl.Dialogs, System.SysUtils, Guitarra;
type

 TGuitarraFactory = class
 public
  function CriarGuitarra(produto: string): IGuitarra; virtual; abstract;
  procedure ExibirDetalhesGuitarra(guitarra: IGuitarra);
 end;

implementation
{ TGuitarraFactory }
 procedure TGuitarraFactory.ExibirDetalhesGuitarra(guitarra: IGuitarra);

 begin
  ShowMessage('Detalhes da guitarra ' +
   guitarra.ObterNomeGuitarra + #13#10#13#10 +
   'Ano Inicial de Fabricação: ' +
   IntToStr(guitarra.ObterAnoInicialFabricao) + #13#10#13#10 +
   'Modelos: ' +
   guitarra.ObterModelos);
 end;
end.
```

Já os tipos TGibsonFactory (**Listagem 7**) e TFenderFactory (**Listagem 8**) estendem a classe TGuitarraFactory, representando fábricas concretas de objetos IGuitarra. Em ambos os casos o método CriarGuitarra foi implementado tomando-se como base um parâmetro ("produto") que indica o instrumento buscado, retornando-se a partir disto uma instância de objeto correspondente.

Implementação da tela para testes do padrão Factory Method

Assim que se concluírem as atividades de codificação das estruturas envolvidas no uso do pattern Factory Method, chega o momento de se implementar a tela a partir da qual serão efetuados os testes. O formulário principal da aplicação terá sua propriedade Name configurada com o valor "FormTeste". A **Listagem 9** apresenta a implementação desta tela.

O método btnObterInformacoesClick define as ações a serem disparadas para o evento OnClick do botão btnObterInformacoes. Através desta operação será empregada uma implementação concreta do tipo TGuitarraFactory (TGibsonFactory ou TFenderFactory), o qual devolverá a partir da invocação do método CriarGuitarra uma instância de IGuitarra.

O uso do pattern Factory Method dispensou a classe TFormTeste da necessidade de conhecer e manipular diretamente as implementações da interface IGuitarra. Levando em conta esse menor grau de dependência em relação a classe concretas, novos tipos derivados de IGuitarra podem ser criados sem que isto leve à necessidade de alterações em TFormTeste.

Executando a aplicação para testes do padrão Factory Method

Ao se disparar a execução do projeto TesteFactoryMethod será exibida inicialmente a tela apresentada na **Figura 3**. Será necessário, a partir disto, selecionar um fabricante e um de seus produtos a fim de obterem informações detalhadas sobre este último.

Acionando-se a opção "Obter Informações" será apresentada uma primeira mensagem que indica que uma instância de IGuitarra foi obtida a partir de uma factory (**Figura 4**). Após isto, uma segunda mensagem apresentará detalhes do instrumento selecionado (**Figura 5**).

Listagem 7. Classe TGibsonFactory

```
unit FactoryGibson;
interface
uses System.SysUtils, Factory, Guitarra, LesPaul, SG;
type
 TGibsonFactory = class(TGuitarraFactory)
  public
   function CriarGuitarra(produto: string): IGuitarra; override;
 end;
implementation
{ TGibsonFactory }
function TGibsonFactory.CriarGuitarra(produto: string): IGuitarra;
begin
  Result:= nil;
  if (UpperCase(produto) = 'LESPAUL') then
    Result:= TGibsonLesPaul.Create
  else if (UpperCase(produto) = 'SG') then
    Result:= TGibsonSG.Create;
end;
end.
```

Listagem 8. Classe TFenderFactory

```
unit FactoryFender;
interface
uses System.SysUtils, Factory, Guitarra, Stratocaster, Telecaster;
type
 TFenderFactory = class(TGuitarraFactory)
  public
   function CriarGuitarra(produto: string): IGuitarra; override;
 end;
implementation
{ TFenderFactory }
function TFenderFactory.CriarGuitarra(produto: string): IGuitarra;
begin
  Result:= nil;
  if (UpperCase(produto) = 'STRATOCASTER') then
    Result:= TFenderStratocaster.Create
  else if (UpperCase(produto) = 'TELECASTER') then
    Result:= TFenderTelecaster.Create;
end;
end.
```

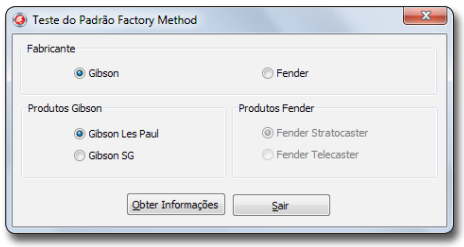


Figura 3. Tela inicial que é exibida ao se executar o projeto TesteFactoryMethod

Utilizando o pattern Template Method

A fim de se construir uma aplicação de exemplo que empregue o padrão Template Method, o primeiro passo será criar um projeto do tipo "VCL Forms Application" chamado TesteTemplateMethod, sendo que o mesmo envolve, de maneira simplificada, a geração de arquivos contendo dados de produtos em formato texto com valores separados pelo caractere "|" ou, ainda, CSV (sigla do inglês comma-separated values, com os valores sendo normalmente separados por

Listagem 9. Implementação do formulário TFormTeste

```
unit FormTesteFactoryMethod;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
  System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms,
  Vcl.Dialogs, Vcl.StdCtrls, Vcl.ExtCtrls;

type
  TFormTeste = class(TForm)
    btnSair: TButton;
    btnObterInformacoes: TButton;
    grpFabricante: TGroupBox;
    rdbFender: TRadioButton;
    rdbGibson: TRadioButton;
    grpFender: TGroupBox;
    rdbStratocaster: TRadioButton;
    rdbTelecaster: TRadioButton;
    grpGibson: TGroupBox;
    rdbLesPaul: TRadioButton;
    rdbSG: TRadioButton;
    procedure btnObterInformacoesClick(Sender: TObject);
    procedure btnSairClick(Sender: TObject);
    procedure rdbGibsonClick(Sender: TObject);
    procedure rdbFenderClick(Sender: TObject);
  private

  public

  end;

var
  FormTeste: TFormTeste;

implementation

  uses Factory, FactoryGibson, FactoryFender, Guitarra;

  {$R *.dfm}

  procedure TFormTeste.rdbGibsonClick(Sender: TObject);
  begin
    rdbLesPaul.Enabled:= rdbGibson.Checked;
    rdbSG.Enabled:= rdbGibson.Checked;
    rdbStratocaster.Enabled:= Not rdbGibson.Checked;
    rdbTelecaster.Enabled:= Not rdbGibson.Checked;
  end;

  procedure TFormTeste.rdbFenderClick(Sender: TObject);
  begin
    rdbStratocaster.Enabled:= rdbFender.Checked;
    rdbTelecaster.Enabled:= rdbFender.Checked;
    rdbLesPaul.Enabled:= Not rdbFender.Checked;
    rdbSG.Enabled:= Not rdbFender.Checked;
  end;

  procedure TFormTeste.btnObterInformacoesClick(Sender: TObject);
  var factoryGuitarras: TGuitarraFactory;
      produto: string;
      instrumento: IGuitarra;
  begin
    if (rdbGibson.Checked) then
    begin
      factoryGuitarras:= TGibsonFactory.Create;
      if (rdbLesPaul.Checked) then
        produto:= 'LESPAUL'
      else
        produto:= 'SG';
    end
    else
    begin
      factoryGuitarras:= TFenderFactory.Create;
      if (rdbStratocaster.Checked) then
        produto:= 'STRATOCASTER'
      else
        produto:= 'TELECASTER';
    end;

    instrumento:= factoryGuitarras.CriarGuitarra(produto);
    ShowMessage('Criada uma instância contendo informações da guitarra:' +
      #13#10#13#10 + instrumento.ObterNomeGuitarra);
    factoryGuitarras.ExibirDetalhesGuitarra(instrumento);
  end;

  procedure TFormTeste.btnSairClick(Sender: TObject);
  begin
    Close;
  end;

end.
```

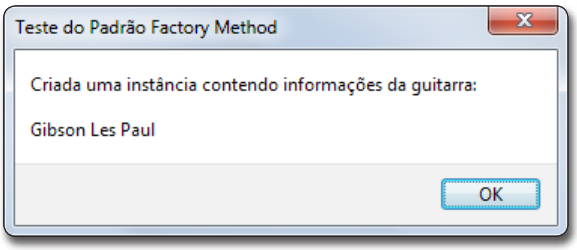


Figura 4. Criação de uma nova instância via factory

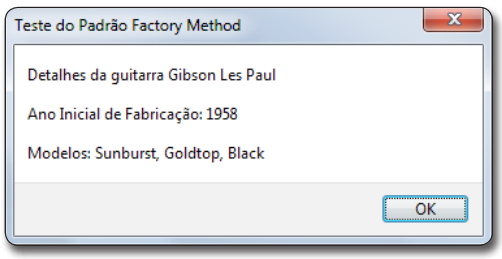


Figura 5. Exibição de informação detalhadas de uma guitarra

Listagem 10. Classe TGeracaoArquivoProdutos

```
unit GeracaoArquivo;

interface

uses System.Classes, System.SysUtils, Datasnap.DBClient, Data.DB;

type
 TGeracaoArquivoProdutos = class
 protected
  function ObterDadosProdutos: TClientDataSet;
  function FormatarLinha(
   codigoBarras, nomeProduto: string;
   preco: Real): String; virtual; abstract;
  procedure SalvarArquivo(stlArquivo: TStringList); virtual; abstract;
 public
  procedure GerarArquivo;
 end;

implementation

{ TGeracaoArquivoProdutos }

 function    .ObterDadosProdutos: TClientDataSet;
var cds: TClientDataSet;
begin
  cds:= TClientDataSet.Create(nil);
  cds.FieldDefs.Add('CdBarraProduto', ftString, 13);
  cds.FieldDefs.Add('NmProduto', ftString, 50);
  cds.FieldDefs.Add('VlProduto', ftCurrency);
  cds.CreateDataSet;

  cds.Append;
  cds.FieldByName('CdBarraProduto').AsString:= '7890000000001';
  cds.FieldByName('NmProduto').AsString:= 'MESA PARA SALA DE JANTAR';
  cds.FieldByName('VlProduto').AsFloat:= 750.98;
  cds.Post;

  cds.Append;
  cds.FieldByName('CdBarraProduto').AsString:= '7890000000002';
  cds.FieldByName('NmProduto').AsString:= 'SOFÁ';
  cds.FieldByName('VlProduto').AsFloat:= 800.5;
  cds.Post;

  cds.Append;
  cds.FieldByName('CdBarraProduto').AsString:= '7890000000003';
  cds.FieldByName('NmProduto').AsString:= 'RACK MOGNO';
  cds.FieldByName('VlProduto').AsFloat:= 687;
  cds.Post;

  Result:= cds;
end;

procedure TGeracaoArquivoProdutos.GerarArquivo;
var cdsDadosProdutos: TClientDataSet;
   stlArquivo: TStringList;
begin
  stlArquivo:= TStringList.Create;

  cdsDadosProdutos:= ObterDadosProdutos;
  cdsDadosProdutos.First;
  while (not cdsDadosProdutos.Eof) do
  begin
   stlArquivo.Add(FormatarLinha(
    cdsDadosProdutos.FieldByName('CdBarraProduto').AsString,
    cdsDadosProdutos.FieldByName('NmProduto').AsString,
    cdsDadosProdutos.FieldByName('VlProduto').AsFloat));
   cdsDadosProdutos.Next;
  end;

  cdsDadosProdutos.EmptyDataSet;
  cdsDadosProdutos.Close;
  FreeAndNil(cdsDadosProdutos);

  SalvarArquivo(stlArquivo);
  stlArquivo.Clear;
  FreeAndNil(stlArquivo);
 end;
end.
```

ponto-e-vírgula). A fonte das informações relativas a tais produtos será, para efeitos de testes, um ClientDataSet temporário.

Criação das classes que empregam o padrão Template Method

Na **Listagem 10** é apresentado o tipo TGeracaoArquivoProdutos. O mesmo é uma aplicação do padrão Template Method, correspondendo a uma classe abstrata em que a operação GerarArquivo representa, basicamente, um algoritmo com os passos para a geração de arquivos contendo informações de produtos. Importante mencionar ainda que GerarArquivo faz uso do método concreto ObterDadosProdutos (responsável por retornar um ClientDataSet temporário), além das operações abstratas FormatarLinha e SalvarArquivo (marcadas com as palavras-chave "virtual" e "abstract") que estarão definidas em subclasses de TGeracaoArquivoProdutos.

A classe TGeracaoArquivoProdutosCSV (**Listagem 11**) corresponde à implementação concreta de TGeracaoArquivoProdutos empregada na geração de arquivos CSV, ao passo que TGeracaoArquivoProdutosTexto (**Listagem 12**) possui a mesma função no que se refere à montagem de arquivos de texto. Em cada um destes casos os métodos FormatarLinha e SalvarArquivo foram implementados seguindo características que são específicas a cada um dos tipos concretos (ambos foram marcados com a palavra chave override).

TGeracaoArquivoProdutosCSV irá gerar um arquivo de nome "teste.csv", já TGeracaoArquivoProdutosTexto fará o mesmo por meio da criação do arquivo "teste.txt". Em ambos os casos a gravação de arquivos é delegada ao método SaveToFile de um objeto do tipo TStringList, com isto acontecendo no mesmo diretório em que se encontrar o executável da aplicação.

Implementação da tela para testes do padrão Template Method

Uma vez que o processo de codificação das classes para a geração de arquivos tenha finalizado, deve-se prosseguir com a implementação da tela que será empregada para os testes das mesmas. O formulário principal da aplicação deve ser renomeado para FormTeste. Na **Listagem 13** está sendo apresentada a implementação do mesmo.

O método btnGerarArquivoClick corresponde ao evento OnClick do botão btnGerarArquivo. A partir do mesmo será utilizada uma implementação concreta da classe TGeracaoArquivoProdutos (TGeracaoArquivoProdutosCSV ou TGeracaoArquivoProdutosTexto), tomando-se como base para isto a opção selecionada em tela por um usuário da aplicação (formato "CSV" ou "Texto").

Listagem 11. Classe TGeracaoArquivoProdutosCSV

```
unit ArquivoCSV;

interface

uses System.Classes, System.SysUtils, GeracaoArquivo;

type
 TGeracaoArquivoProdutosCSV = class(TGeracaoArquivoProdutos)
 protected
  function FormatarLinha(
   codigoBarras, nomeProduto: string;
   preco: Real): String; override;
  procedure SalvarArquivo(stlArquivo: TStringList); override;
 end;

implementation

{ TGeracaoArquivoProdutosCSV }

function TGeracaoArquivoProdutosCSV.FormatarLinha(codigoBarras,
 nomeProduto: string; preco: Real): String;
begin
 Result:= codigoBarras + ';' + nomeProduto + ';' +
  FloatToStr(preco);
end;

procedure TGeracaoArquivoProdutosCSV.SalvarArquivo(stlArquivo: TStringList);
begin
 stlArquivo.SaveToFile('teste.csv');
end;

end.
```

Listagem 12. Classe TGeracaoArquivoProdutosTexto

```
unit ArquivoTexto;

interface

uses System.Classes, System.SysUtils, GeracaoArquivo;

type
 TGeracaoArquivoProdutosTexto = class(TGeracaoArquivoProdutos)
 protected
  function FormatarLinha(
   codigoBarras, nomeProduto: string;
   preco: Real): String; override;
  procedure SalvarArquivo(stlArquivo: TStringList); override;
 end;

implementation

{ TGeracaoArquivoProdutosTexto }

function TGeracaoArquivoProdutosTexto.FormatarLinha(codigoBarras,
 nomeProduto: string; preco: Real): String;
begin
 Result:= '|' + codigoBarras + '|' + nomeProduto + '|' +
  FloatToStr(preco) + '|';
end;

procedure TGeracaoArquivoProdutosTexto.SalvarArquivo(stlArquivo: TStringList);
begin
 stlArquivo.SaveToFile('teste.txt');
end;

end.
```

A definição das funcionalidades gerais para o processo de geração de arquivos TGeracaoArquivoProdutos permitiu, por sua vez, que as subclasses TGeracaoArquivoProdutosCSV e TGeracaoArquivoProdutosTexto apenas se preocupassem com a implementação de aspectos específicos em suas respectivas estruturas. Tudo isso atesta o foco no reuso que o padrão Template Method tanto enfatiza.

Executando a aplicação para testes do padrão Template Method

Ao se executar o projeto TesteTemplateMethod será exibida uma tela como a indicada na **Figura 6**, na qual deverá ser selecionada alguma das opções para a geração de arquivos ("CSV" ou Texto).

Caso se opte pela criação de um arquivo .csv, o usuário irá se deparar com a janela apresentada na **Figura 7**. Já a **Figura 8** apresenta o arquivo .csv que foi gerado e, neste caso, carregado a partir do Excel.

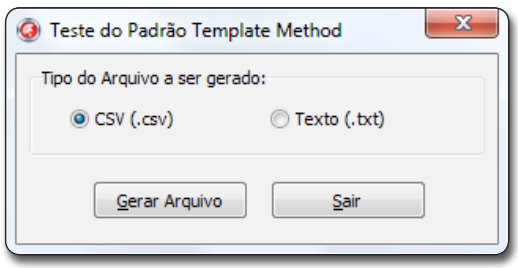


Figura 6. Tela inicial que é exibida ao se executar o projeto TesteTemplateMethod

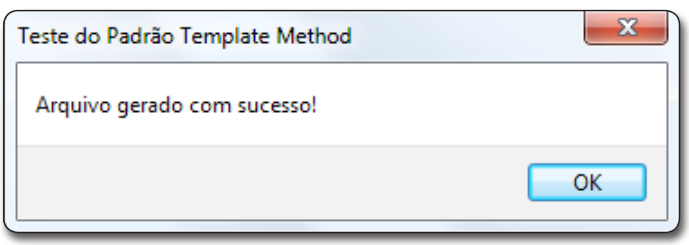


Figura 7. Aviso de que o arquivo .csv foi gerado com sucesso

Listagem 13. Implementação do formulário TFormTeste

```
unit FormTesteTemplateMethod;
interface
uses
 Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.
Classes, Vcl.Graphics,
 Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Data.DB, Vcl.StdCtrls, Vcl.ExtCtrls;
type
 TFormTeste = class(TForm)
  btnGerarArquivo: TButton;
  grpTipoArquivo: TRadioGroup;
  btnSair: TButton;
  rdbArquivoCSV: TRadioButton;
  rdbArquivoTexto: TRadioButton;
  procedure btnGerarArquivoClick(Sender: TObject);
  procedure btnSairClick(Sender: TObject);
 private
 public
 end;
var
 FormTeste: TFormTeste;
 stlArquivo: TStringList;
implementation
 uses GeracaoArquivo, ArquivoCSV, ArquivoTexto;
{$R *.dfm}
 procedure TFormTeste.btnGerarArquivoClick(Sender: TObject);
 var objGeracaoArquivo: TGeracaoArquivoProdutos;
 begin
  if (rdbArquivoCSV.Checked) then
   objGeracaoArquivo:= TGeracaoArquivoProdutosCSV.Create
  else
   objGeracaoArquivo:= TGeracaoArquivoProdutosTexto.Create;
  objGeracaoArquivo.GerarArquivo;
  ShowMessage('Arquivo gerado com sucesso!');
 end;
 procedure TFormTeste.btnSairClick(Sender: TObject);
 begin
  Close;
 end;
end.
```

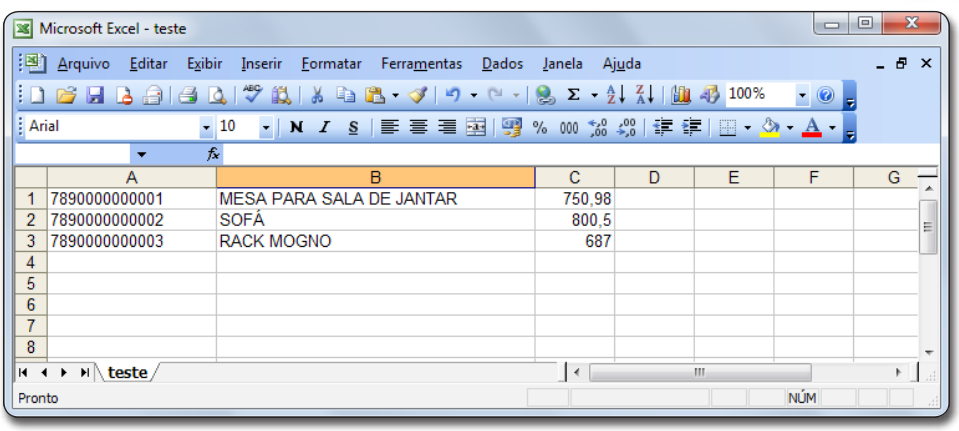


Figura 8. Arquivo teste.csv

A geração de um arquivo-texto segue um processo similar ao mesmo que foi especificado para aqueles de extensão .csv. Na **Figura 9** está um exemplo disto.

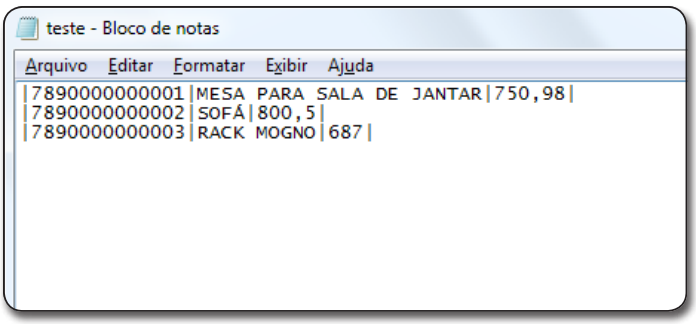


Figura 9. Arquivo teste.txt

Conclusão

Conforme discutido neste artigo, design patterns são soluções que podem ser empregadas em determinados tipos de situações dentro do desenvolvimento de software. Diversos padrões foram concebidos no decorrer do tempo, com cada um destes sendo aplicáveis dentro de um conjunto de condições específicas.

O uso do padrão Factory Method pode contribuir para um menor acoplamento entre partes de uma aplicação, haja vista a diminuição da dependência entre classes que o mesmo proporciona por incentivar o uso de interfaces e abstrações. Além disso, este pattern centraliza a lógica de criação de objetos derivados de um tipo básico, evitando assim a duplicação de código ao longo de um projeto e, conseqüentemente, possíveis dificuldades futuras na manutenção da solução.

Já o pattern Template Method tem por finalidade principal, basicamente, fornecer meios para o tratamento de comportamentos variáveis dentro de uma sequência de operações (algoritmo). A lógica para o processamento de uma determinada ação permanece restrita a um tipo abstrato, ao passo que aspectos específicos são implementados por subclasses do mesmo. Logo, conclui-se que a principal vantagem de se empregar este padrão está na ênfase que o mesmo dá ao reuso de código.

A escolha por quais patterns implementar dentro de uma solução dependerá sempre das características desta última. Uma análise detalhada auxiliará na escolha das melhores alternativas, de maneira que isto não apenas contribua para a obtenção de um software bem estruturado, como também tornará a aplicação em questão mais flexível diante da necessidade de mudanças.

Renato José Groffe
renato.groffe@yahoo.com.br
Atua como consultor em atividades voltadas ao desenvolvimento de softwares há mais de 10 anos. Bacharel em Sistemas de Informação, com especialização em Engenharia de Software. Microsoft Certified Technology Specialyst (Web, WCF, Distributed Applications, ADO.NET, Windows Forms), Oracle Certified Associate (PL/SQL), Sun Certified (SCJP, SCWCD), ITIL Foundation V2.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

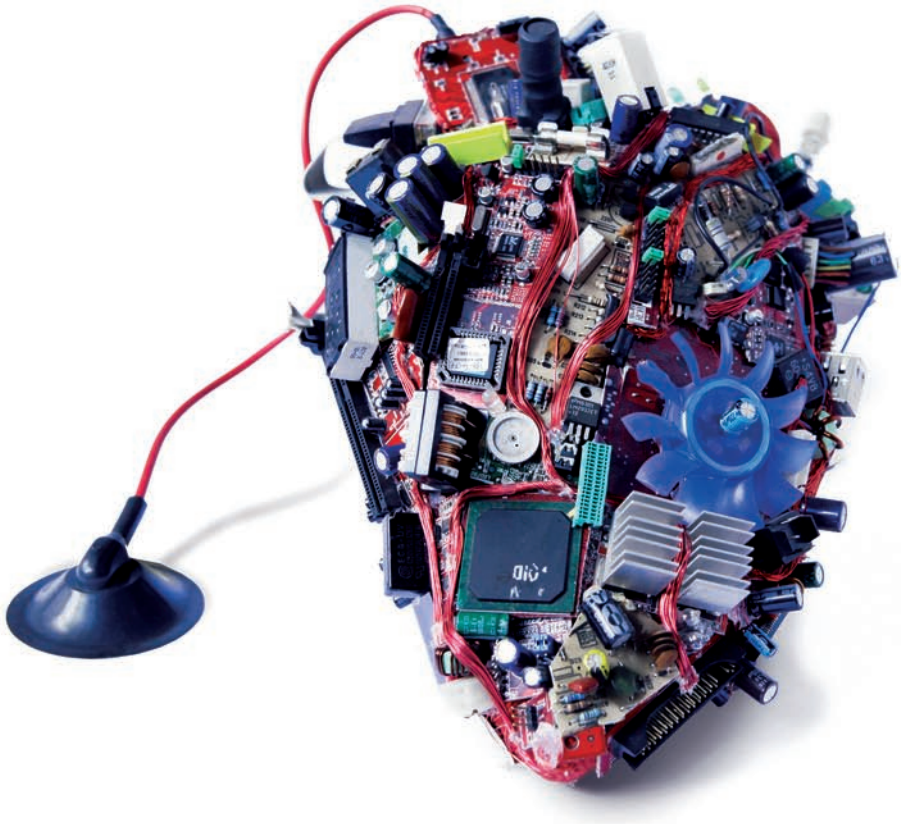
DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

O que o Angry Birds, empreendedorismo e um matemático têm em comum?



Todos fazem parte do maior acontecimento de tecnologia e internet do mundo.

Campus Party™

06 a 12 de fevereiro de 2012

Acompanhe a programação ao vivo pela Campus TV.
www.campus-party.com.br

Patrocinador Principal:

Telefônica | vivo

Apoio:

anhembi parque
www.anhembi.com.br

Assista ao vídeo e descubra mais sobre a Campus Party Brasil